ON THE DESIGN OF VLSI CIRCUITS FOR THE WINOGRAD FOURIER TRANSFORM ALGORITHM (U)

by

Pierre Lavoie and Serge Martineau

DEFENCE RESEARCH ESTABLISHMENT OTTAWA
REPORT NO. 1108

Canada

cember 1991
Ottawa

National Defence / Défense nationale

ON THE DESIGN OF VLSI CIRCUITS FOR THE WINOGRAD FOURIER TRANSFORM ALGORITHM (U)

by

Pierre Lavoie and Serge Martineau
Radar ESM Section
Electronic Warfare Division

DEFENCE RESEARCH ESTABLISHMENT OTTAWA
REPORT NO. 1108

PCN
011LB

December 1991
Ottawa

ABSTRACT

A VLSI architecture for computing the discrete Fourier transform (DFT) using the Winograd Fourier transform algorithm (WFTA) is presented. This architecture is an addressless, routed, bit-serial scheme that directly maps an N-point algorithm onto silicon. The architecture appears to be far less costly than systolic schemes for implementing the WFTA, and faster than current FFT devices for similar transform sizes. The nesting method of Winograd is used for partitioning larger transformations into several circuits. The advantage of this partitioning technique is that it allows using circuits that are all of the same type. However, the number of input/output pins of each circuit is higher than with some other approaches like, for example, the prime factor algorithm. The design of a 20-point DFT circuit with logic diagrams of its major cells is presented. The gate array circuit has been sent for fabrication in a 0.7μm CMOS technology. Five circuits interconnected together will compute 60-point complex transforms at a rate of one transformation every 0.53μs.

RÉSUMÉ

Une architecture VLSI pour le calcul de la transformation discrète de Fourier en utilisant l'algorithme de transformation de Fourier de Winograd (ATFW) est présentée. Cette architecture est un arrangement routé, bit-sériel, et sans adresses qui transpose directement un algorithme de taille donnée sur silicium. L'architecture s'avère être beaucoup moins coûteuse que les systèmes systoliques pour implanter l'ATFW, et plus rapide que les dispositifs courants de transformation rapide de Fourier pour des longueurs de transformation comparables. La méthode de "tissage" de Winograd est utilisée pour fragmenter des transformations plus longues sur plusieurs circuits distincts. L'avantage de cette technique de fragmentation est qu'elle permet d'utiliser des circuits tous du même type. Cependant, le nombre de broches d'entrée/sortie de chaque circuit est plus élevé qu'avec d'autres approches comme, par exemple, l'algorithme de factorisation premier. La conception d'un circuit de transformation discrète de Fourier pour 20 points est présentée, avec des diagrammes logiques pour ses principales cellules. Le circuit a été soumis pour fabrication avec une matrice de portes dans un procédé CMOS de 0.7μm. Cinq circuits interconnectés ensemble pourront calculer des transformations complexes de 60 points à une vitesse de une transformation à toutes les 0.53μs.

EXECUTIVE SUMMARY

Electronic warfare systems rely more and more on the development of digital processing to increase their signal handling capability. This trend stems in good part from the convenience and low cost of semiconductor devices and the emergence of very large scale integration (VLSI) systems. The object of this research is to investigate new means of computing the *discrete Fourier transform (DFT)* at very high speed using VLSI circuits. The discrete Fourier transformation is a widely used algorithm for switching between the time and frequency representations of sampled waveforms.

The performance of DFT circuits and boards is determined by the transformation algorithm and by the architecture used to implement the algorithm. In commercial products, the algorithm and architecture are chosen for their flexibility, in an attempt to facilitate many applications. Practically all commercial devices use the FFT algorithm, which allows varying the transform size N across a wide range of values and implementing the division by N of the inverse transformation with an inexpensive bit shift.

Unfortunately, commercial DFT circuits and boards don't deliver the throughputs that are needed in many electronic warfare applications. Higher throughputs can be obtained by using several DFT processors in parallel, but this generally leads to complicated and expensive implementations, which are limited by the need for multiplexers and demultiplexers, increased bulk, lower reliability, and higher power consumption.

This report presents a DFT architecture aimed at applications where DFTs must be computed at very high speeds, and where the number of points N is fixed and not too large, typically a few hundreds or less. This architecture is not based on the FFT. Instead it uses an algorithm that was invented by Winograd in 1976. The *Winograd Fourier transform algorithm (WFTA)* computes the same transformation as the FFT, and uses fewer multiplications. In the proposed architecture, an N-point WFTA is mapped directly onto a VLSI circuit using an addressless, bit-serial scheme. The smaller number of multiplications yields silicon area savings which can be traded for a higher throughput or a larger transform size N. Due to the complicated indexing scheme of the WFTA, the layout requires substantial routing between its arithmetic cells and the architecture is called *routed*.

If a layout turns out to be too large to fit on a single circuit, the architecture can be partitioned into several identical circuits using the *nesting method*. The nesting method has also been invented by Winograd. As higher length algorithms are constructed, the nesting method uses less multiplications than other construction algorithms, including the prime factor algorithm.

To validate the routed architecture and the partitioning strategy, a VLSI circuit has been designed at DREO and sent to a silicon foundry. The CMOS circuit contains 55 000 gates and can compute by itself 20-point complex DFTs. The nesting method allows the interconnection of five circuits to compute 60-point complex DFTs. Assuming 16-bit input samples, the predicted speed of 1.8 million transformations per second is about three to ten times higher than that of commercial chip sets*. In the prototype circuit, the adders have been organized in layers and interconnected by software. The 48 multipliers have been carefully designed to minimize their gate count without compromising their speed and accuracy. The circuit can accept samples of any precision in fixed-point two's complement format, and output coefficients with up to 10 bits of accuracy.

The Air Force Institute of Technologyin in Dayton, Ohio, is also developing Winograd Fourier transform circuits. At this time, 15-, 16-, and 17-point DFT circuits are being designed and tested. The three full-custom circuits are slightly slower than the DREO gate array, but they are more accurate because their multipliers have more stages. The three circuits are meant to be interconnected using the prime factor algorithm to form part of a 4080-point DFT machine†.

The WFTA and the routed architecture are not without disadvantages. First, as the transform size is increased, the routing gradually grows and may become impractical to handle. Second, the complex indexing scheme of the WFTA restrains the flexibility with respect to N. Lastly, the WFTA favors using values of N that are not powers of 2; hence the division by N appearing in the inverse transformation does not reduce to a bit shift like in the FFT.

The WFTA should not be viewed as a replacement for the FFT, but rather as a complementary algorithm with its own advantages and inconveniences, which may find use in different applications. The WFTA and the routed architecture are attractive for applications requiring high throughput, cost effective DFT computation for moderate transform sizes. For instance, the routed architecture is currently being considered at DREO for processing radar pulses in real-time upon interception by radar electronic support measures (ESM) systems.

*The fastest available chip set, to the authors' knowledge, is manufactured by Honeywell and consists of 12 GaAs circuits.

†The authors whish to thank Mark A. Mehalic, AFIT, for the information provided.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1.0 INTRODUCTION

The introduction by Winograd [1],[2], and Agarwal and Cooley [3], of new, short length discrete Fourier transformation algorithms requiring fewer multiplications than the fast Fourier transform [4],[5] stirred interest in the signal processing community. In combination with his high speed algorithms, Winograd proposed a nesting method for constructing algorithms of higher lengths. The algorithms obtained by means of this nesting method are known as *Winograd Fourier transform algorithms (WFTA)* [6].

Another method, which is based on the Good-Thomas prime factor algorithm (PFA) [7],[8], has been proposed by Kolba and Parks [9] for computing long discrete Fourier transforms with Winograd's short length algorithms. The PFA and nesting methods can be combined, making it possible to obtain in-place and in-order algorithms [10]. However, only the nesting method of Winograd is considered in this report, mostly because it minimizes the number of multiplications.

Apart from their theoretical value, which was immediately recognized, Winograd's algorithms have found very few applications since their introduction. One of the underlying difficulties with these algorithms is that their additions are nested in a complicated and irregular manner. Early results showed that WFTA software sometimes runs faster, or slower, than the FFT on computers like the IBM 370 [6],[9],[11]-[14]. Various hardware architectures for the WFTA and its variants have been proposed, but, to the authors' knowledge, none has been demonstrated using a complete prototype. As a result, the FFT is still considered as the algorithm of choice in the practical world.

The widespread perception that Winograd's algorithms do not lend themselves well to either hardware or software realizations is now being challenged. The change stems from the emergence of new computer architectures, higher component densities on VLSI circuits, and more powerful compilers and computer-aided design tools. For instance, Lu, Cooley and Tolimieri have recently shown [15] that variants of the WFTA can execute more efficiently than the FFT on RISC computers having a "floating-point multiply-add" feature. Aloisio *et al.* have implemented the PFA on hypercube computers [16]. At this time, the Air Force Institute of Technology is developing 15-, 16-, and 17-point WFTA integrated circuits [17],[18][1].

The argument for using the WFTA instead of the FFT in high-speed VLSI realizations is simple. Since the WFTA requires fewer multiplications than the FFT, and

[1] After work on the present report was well under way, the authors became acquainted with the AFIT project and were pleasantly surprised by the similarities between the AFIT and DREO circuits.

multipliers in VLSI are very expensive[2], the WFTA should yield smaller, more cost-eff ve VLSI realizations. Figure 1 shows the minimum number of multiplications in the

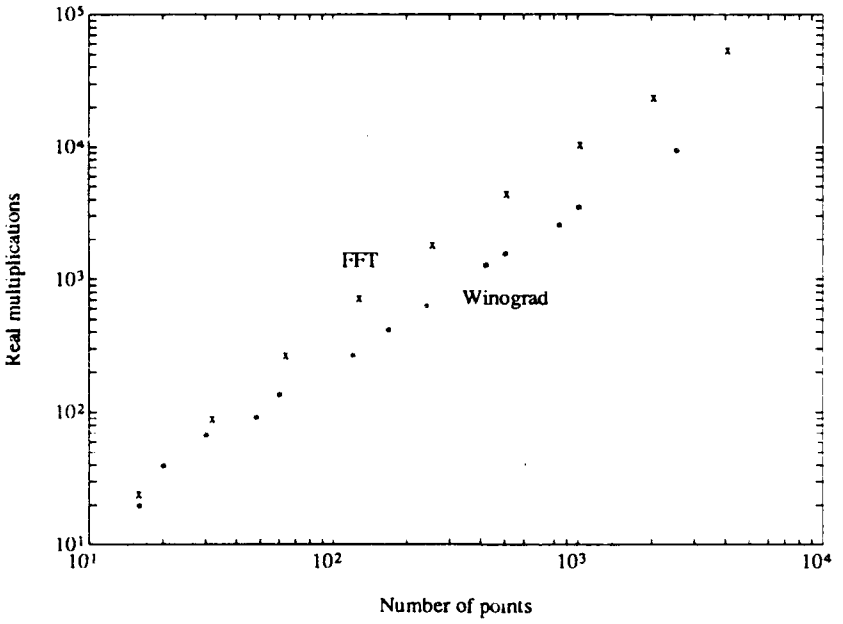


Figure 1: Number of non-trivial real multiplications in the FFT and Winograd algorithms as a function of the number of points N.

WFTA and FFT[3], as a function of the number of points N. It is easily verified that the WFTA requires two to three times fewer multiplications than the FFT for $N > 60$. The difference increases with N, as the WFTA requires a number of multiplications proportional to N, while for the FFT the proportionality is to $N \log(N)$ [20]. The number of additions remains approximately the same.

In this report, we examine the implementation of the WFTA in VLSI form for the high-speed calculation of moderate length (less than a few hundred points) discrete Fourier transforms. We propose a new VLSI architecture with detailed designs of its different hardware cells. To put this in perspective, a quick review of some of the architectures proposed in the past is useful. In 1980, Zohar [21] proposed running the WFTA on a dedicated, address-based machine with one multiplier and two adders. In 1982, Ward and

[2]A CMOS multiplier of length l_b bits typically contains about $2\,l_b$ times more gates than an adder.

[3]In Fig. 1, the number of arithmetic operations in the radix-2 FFT has been reduced by exploiting the symmetries in the sine and cosine functions, and by implementing the complex multiplications with three real multiplications and three real additions. The complex multiplication algorithm with three real multiplications can be found in [19, Sect. 3.7.2].

Stanier [22] designed a systolic architecture for the WFTA; such architectures produce regular layouts and allow very high clock rates [23],[24]. Then, in 1983, MacLeod and Bragg [25] suggested directly mapping the algorithm's data flow onto hardware, using bit-serial arithmetic. In 1985, Costello [26] compared the PFA to other techniques for radar beam forming and concluded that the former was much cheaper to implement with dedicated hardware. At about the same time, Ward, McCanny, and McWhirter [27],[28], and shortly after, Owens and Ja'Ja [29], introduced more systolic architectures for the WFTA. Lastly, in 1988, Linderman *et al.* [18] presented the design of three full-custom WFTA circuits destined for a 4080-point PFA realization. The operations in the circuits are carried out bit-serially, while the data transfers between the circuits and the main memory are bit-parallel.

Pursuing the idea of MacLeod and Bragg, we propose an addressless, bit-serial architecture that directly maps the WFTA of interest onto a VLSI circuit. Probing further, we examine in detail the hardware cells and their interconnections, and actually provide the specifications of a 20-point WFT circuit that has been sent for fabrication in a 0.7 μm "gate array" CMOS technology. We found that for implementing the WFTA, an approach like MacLeod and Bragg's yields the same performance as a systolic architecture, but at a lower cost [30]. A 20-point WFT circuit, for example, contains about 47 000 gates and fits on a moderately large gate array. By comparison, the systolic architecture of Ward *et al.* would require 300 000 gates[4] and a much larger die size. Hence, the layout of our circuit ends up being more compact, despite some irregular portions having complicated routing. From a design effort standpoint, the 20-point WFT circuit's schematics were manually entered in the chip manufacturer's design system in seven man-weeks. Interconnecting the adders took only a small portion of that time. As early as next year, some chip manufacturers will add to their design software sets a "logic synthesis" tool that will directly read logic specifications, and eliminate the error-prone and often tedious task of drawing the schematics[5]. This will make routed architectures more attractive in general.

The novelty of the proposed architecture lies at the system level, where the WFT circuits exchange data for computing discrete Fourier transforms of higher lengths. Instead of relying on the standard PFA for partitioning the transformation, we took Winograd's nesting method. This allowed us to design a 20-point WFT circuit such that by assembling five devices, they can compute 60-point transforms[6]. Including this feature

[4] See Section 7.0 for gate count equations.

[5] The authors are grateful to A. Boubguira, LSI Logic Co. of Canada, for the information provided.

[6] Another possibility, in fact, would be to use a single device five times in succession. This discussion ignores the possible reduction in hardware which can be obtained for lower transformation rates.

in the 20-point WFT circuit increased the gate count by 16%[7] and added 80 pins to the package. The advantage of this approach is that it can be implemented with circuits all of the same type. Also, the number of multipliers that are used is always kept to a minimum, thus as higher density processes become available, a multi-circuit configuration can be directly combined to fit on a single integrated circuit. The disadvantage of the approach is that it requires more pins than the PFA for inter-circuit data exchanges. This is the price for minimizing the number of multipliers in the data path and for using circuits that are identical.

Table 1: Comparison of the proposed WFTA architecture to commercial FFT devices..

Device(s)	Circuit Count	Clock Rate	N	Throughput (samples/s)	Figure of Merit
L64280/81 (LSI Logic)* [31]	3	40MHz	64	4.3×10^6	1.4
A41102 (Austek Microsystems) [32]	1	40MHz	64	2.5×10^6	2.5
HFFP (Honeywell)†	12	250MHz	64	41.7×10^6	3.5
a66110/210 (array Microsystems) [33]	2	40MHz	64	13.1×10^6	6.6
PDSP16510 (Plessey Semicond.) [34]	1	40MHz	64	16.4×10^6	16.4
WFT circuit (DREO)	5	30MHz	60	111.1×10^6	22.2

* This is a floating point chip set. All the others are fixed point.

† This is a GaAs chip set (advance information 10/91). All the others are CMOS.

For comparing the proposed WFTA architecture to current FFT schemes, we use four FFT chip sets that are commercially available[8]. It is assumed that the discrete Fourier transformations are on complex data. Table 1 gives the number of circuits in each set, the clock rate, and the throughput rate in complex samples/s for a transformation of length N. Also shown is a simple but intuitive "figure of merit" obtained by dividing the throughput by the number of circuits. The higher the figure of merit is, the better. Of the FFT circuits, the PDSP16510 by Plessey Semiconductors is the only one that has a figure of merit close to that of the WFT circuit. However, the PDSP16510 contains at least twice as many gates as the WFT circuit, and costs about four times as much. Compensating for silicon area and speed discrepancies would increase the figure of merit of the WFT circuit to 75, *i.e.* at about four times the value of the top FFT device[9].

[7] The gate count of the circuit therefore adds up to 47 000 + 8 000 = 55 000 gates.

[8] The FFT can also be computed on digital signal processors [35], but at slower speeds.

[9] To obtain the higher figure of merit, one could fit the 175 000 gates required by the 60-point WFTA on two larger or higher-density circuits and raise their clock rate to 40MHz.

On the other hand, the FFT devices offer more flexibility with regards to the number of points of the transformation. The WFT circuit is limited to two transformation sizes: 20 and 60 points[10]. This comparison illustrates well that the FFT and WFTA offer different advantages and limitations, and are therefore suited to different applications.

This report is addressed to scientists who are studying the high-speed calculation of the discrete Fourier transform, to engineers who design hardware for that computation, *i.e.* VLSI circuits, and possibly to users of this hardware. No specific mathematical background is required. The hardware descriptions are very detailed, mostly because the only way to obtain accurate gate counts, and cost estimates, is by unfolding a complete logic design. It is our hope that the logic cells presented here will be helpful in other bit-serial circuits. S. Martineau did most of the cell design work. P. Lavoie proposed the VLSI architecture and compared it to other schemes from speed and cost standpoints.

The report is organized as follows. In the next section, the discrete Fourier transformation, Winograd's short length algorithms and the systolic architecture of Ward, McCanny, and McWhirter are briefly reviewed. The routed architecture is introduced in Section 3.0 using a 5-point transformation example. Then, in Section 4.0, the logic design of a 20-point WFTA circuit based on the routed architecture is presented in detail. A technique for laying out and interconnecting the adders is proposed. Multipliers with small gate counts are introduced. The partitioning of a higher length 60-point algorithm in five circuits is explained. This partitioning follows a novel approach based on Winograd's nesting method. Internal scaling of the data to prevent overflows is included in all the cells. The section ends with a recapitulation of the various cells that are required and their gate counts. In Section 5.0, computer simulations of the 20-point WFTA circuit are presented. The accuracy of the Fourier coefficients produced by the circuit is measured. Practical considerations like the testability, clock speed and pin requirement are examined in Section 6.0. Lastly, in Section 7.0, the routed architecture is compared to the systolic architecture of Ward *et al.* and to a straightforward FFT design from a cost point of view. The 20- and 60-point WFTA algorithms are derived in Appendix A. The twiddle factors that must be stored into the WFT circuit are listed in Appendix B. The logic symbols appearing in the figures are described in Appendix C.

[10] In many high speed applications this is not much of an inconvenience since the number of points N is fixed.

2.0 WARD, McCANNY AND McWHIRTER'S SYSTOLIC ARCHITECTURE

2.1 THE DISCRETE FOURIER TRANSFORMATION

The N-point discrete Fourier transform synthesis equation is

$$A_k = \sum_{n=0}^{N-1} a_n W^{nk} , \quad k = 0, 1, \ldots, N-1 , \tag{1}$$

with

$$W = e^{-j(2\pi/N)} , \tag{2}$$

where $\{A_0, A_1, \ldots, A_{N-1}\}$ denotes the *discrete Fourier transform (DFT)* of a sequence of N evenly-spaced, possibly complex samples $\{a_0, a_1, \ldots, a_{N-1}\}$. The operation of computing the DFT of a sequence is called the *discrete Fourier transformation.* The original sequence can be recovered from its DFT by the analysis equation

$$a_n = \frac{1}{N} \sum_{k=0}^{N-1} A_k W^{-nk} , \quad n = 0, 1, \ldots, N-1 . \tag{3}$$

This operation is called the *inverse discrete Fourier transformation.* It is very similar in form to the discrete Fourier transformation.

The inverse discrete Fourier transformation can be implemented using a forward DFT device by reversing the order of the outputs 1 through $N-1$ and dividing their value by N. If $N = q^r$ and numbers are represented in q-ary digits, then the division by N reduces to shifting the point r positions. When using an FFT, N is usually a power of two, and inverse transforms are thus easily computed. Winograd algorithms for $N = 2^r$ have been derived [36]-[38], but they require more multiplications and additions than the FFT for $N \geq 32$.

2.2 WINOGRAD'S SHORT LENGTH DISCRETE FOURIER TRANSFORMATION ALGORITHMS

Each of the short length discrete Fourier transformation algorithms introduced by Winograd consists of a sequence of additions, followed by multiplications, and by more additions. Winograd has given algorithms for 2-, 3-, 4-, 5-, 7-, 8-, 9-, and 16-point DFTs, and algorithms for other lengths can be found in [36]-[40].

For example, the 5-point algorithm producing the DFT $\{A_0, A_1, A_2, A_3, A_4\}$ of an input sequence $\{a_0, a_1, a_2, a_3, a_4\}$ consists of the following operations:

Additions:

$$s_1 = a_1 + a_4 \quad s_2 = a_1 - a_4 \quad s_3 = a_3 + a_2 \quad s_4 = a_3 - a_2$$
$$s_5 = s_1 + s_3 \quad s_6 = s_1 - s_3 \quad s_7 = s_2 + s_4 \quad s_8 = s_5 + a_0$$

Multiplications:

$$\left.\begin{array}{ll} m_0 = 1 \cdot s_8 & m_1 = \left(\frac{\cos u + \cos 2u}{2} - 1\right) \cdot s_5 \\ m_2 = \left(\frac{\cos u - \cos 2u}{2} - 1\right) \cdot s_6 & m_3 = i(\sin u + \sin 2u) \cdot s_2 \\ m_4 = i \sin 2u \cdot s_7 & m_5 = i(\sin u - \sin 2u) \cdot s_4 \end{array}\right\} \text{ with } u = -\frac{2\pi}{5}$$

Additions:

$$s_9 = m_0 + m_1 \quad s_{10} = s_9 + m_2 \quad s_{11} = s_9 - m_2 \quad s_{12} = m_3 - m_4$$
$$s_{13} = m_4 + m_5 \quad s_{14} = s_{10} + s_{12} \quad s_{15} = s_{10} - s_{12} \quad s_{16} = s_{11} + s_{13}$$
$$s_{17} = s_{11} - s_{13}$$

Output:

$$A_0 = m_0 \quad A_1 = s_{14} \quad A_2 = s_{16} \quad A_3 = s_{17} \quad A_4 = s_{15} \quad .$$

The algorithm contains 17 complex additions[11] and 6 complex multiplications. The fixed factor in each multiplication is either purely real or imaginary, never a complex number. This is a property of Winograd's algorithms. Each complex multiplication can therefore be computed using just two real multiplications instead of three or four.

2.3 WARD, McCANNY AND McWHIRTER'S SYSTOLIC ARCHITECTURE

The architecture presented here has been proposed by Ward, McCanny and McWhirter [27],[28]. It has been chosen over other architectures [22],[29] because it is complete, simple, and representative of the group. This architecture falls into the "systolic" category, and hence it has several attractive properties [23],[24]. However it is rather inefficient in its use of the silicon area, especially for large transform sizes. This drawback actually motivated us to develop a different, more compact architecture, which is presented in the next section. Since the systolic architecture provides general insight into the implementation of Winograd's algorithms, and might be useful for some applications, it is described first.

In the systolic architecture, the number of multipliers is kept small, whereas the number of adders is allowed to grow proportionally to N^2. This allows computing the

[11] The term *additions* as used here refers to both the addition and the subtraction operations of the algorithm.

additions using four regular arrays of cells. For implementing an N-point transformation, the "systolic" arrays are programmed in such a way that some cells are active, *i.e.* compute an addition, while others are simply used as delay units. The arrays compute far more additions than required by Winograd's algorithms, and the resultant layouts are therefore not as compact as they could be. On the other hand, since the layout regularity is very high, both the design time and the risk of a design error are reduced. The clock rate of the arrays is independent of their size, and very high computational speeds can be reached even with very large arrays [24].

The architecture accepts and processes the N input samples in parallel, and is called *bit-serial* because the samples enter and travel in the circuit in a serial fashion, *i.e.* bit by bit. The data rate is inversely proportional to the number of bits per sample. High processing rates are achieved by computing all the N Fourier coefficients in parallel. The architecture requires $2N$ input and $2N$ output pins, for the complex data samples and Fourier coefficients, respectively.

The architecture is best understood when an N-point transformation is written in the form:

$$\mathbf{A} = \mathbf{Z}\,(\mathbf{Xa} \times \mathbf{Yb})\,. \tag{4}$$

The input samples and DFT coefficients form the vectors $\mathbf{a}$ and $\mathbf{A}$, respectively. $\mathbf{X}$ is an $M \times N$ (row $\times$ column) matrix and $\mathbf{Z}$ is an $N \times M$ matrix, where M denotes the number of complex multiplications of the N-point transformation. The matrices $\mathbf{X}$ and $\mathbf{Y}$ contain only $+1$, -1 and 0 values. It is through the matrix-vector products that the additions of Winograd's algorithm are carried out. The product $\mathbf{Yb}$ can be precalculated so as to form a set of M *twiddle factors* [12] having either a purely real or imaginary value.

The 5-point transformation [2], for example, can be calculated using (4) where

$$\mathbf{X} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & -1 & -1 & 1 \\ 0 & 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 1 & -1 \\ 0 & 0 & -1 & 1 & 0 \end{pmatrix},$$

[12] These twiddle factors play a role similar to the twiddle factors in the FFT, but they differ from the latter in number and value.

$$(\mathbf{Yb})^t = \begin{pmatrix} 1 \\ \left(\frac{\cos u + \cos 2u}{2} - 1\right) \\ \left(\frac{\cos u - \cos 2u}{2} - 1\right) \\ i(\sin u + \sin 2u) \\ i \sin 2u \\ i(\sin u - \sin 2u) \end{pmatrix} \quad \text{with } u = \frac{2\pi}{5} ,$$

and

$$\mathbf{Z} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & -1 & 0 \\ 1 & 1 & -1 & 0 & 1 & 1 \\ 1 & 1 & -1 & 0 & -1 & -1 \\ 1 & 1 & 1 & -1 & 1 & 0 \end{pmatrix} .$$

The systolic architecture would yield an implementation with $2M = 12$ multipliers, and $4MN = 120$ array cells, of which 84 would be performing additions. When considering that Winograd's algorithm uses just 34 additions, the systolic architecture appears inefficient. However, it is simple and very regular.

The number of gates in one array cell is now examined. Using the cell functionality described in [27], a logic diagram, such as the one shown in Fig. 2, can be designed. This particular design contains 84 gates if it were implemented using a popular CMOS library [41] with flip-flops featuring clear and scan. Based on this design, the total number G_a of gates in the four systolic arrays is:

$$G_a = 336\,NM\ . \tag{5}$$

The main drawback of this architecture is that the systolic arrays requires a number of cells that is proportional to N^2. As a result, the architecture quickly becomes prohibitively expensive as N increases.

3.0 A ROUTED ARCHITECTURE FOR THE WFTA

In this section, a cost-effective bit-serial architecture for the WFTA is presented. Even though this architecture lacks some of the elegant properties of the systolic architectures [24], it should yield circuits having smaller areas, and hence allow discrete Fourier transformations of higher lengths.

In the proposed architecture, an N-point WFTA is mapped directly onto silicon, with a minimum of modifications. This follows the idea of MacLeod and Bragg [25]. The

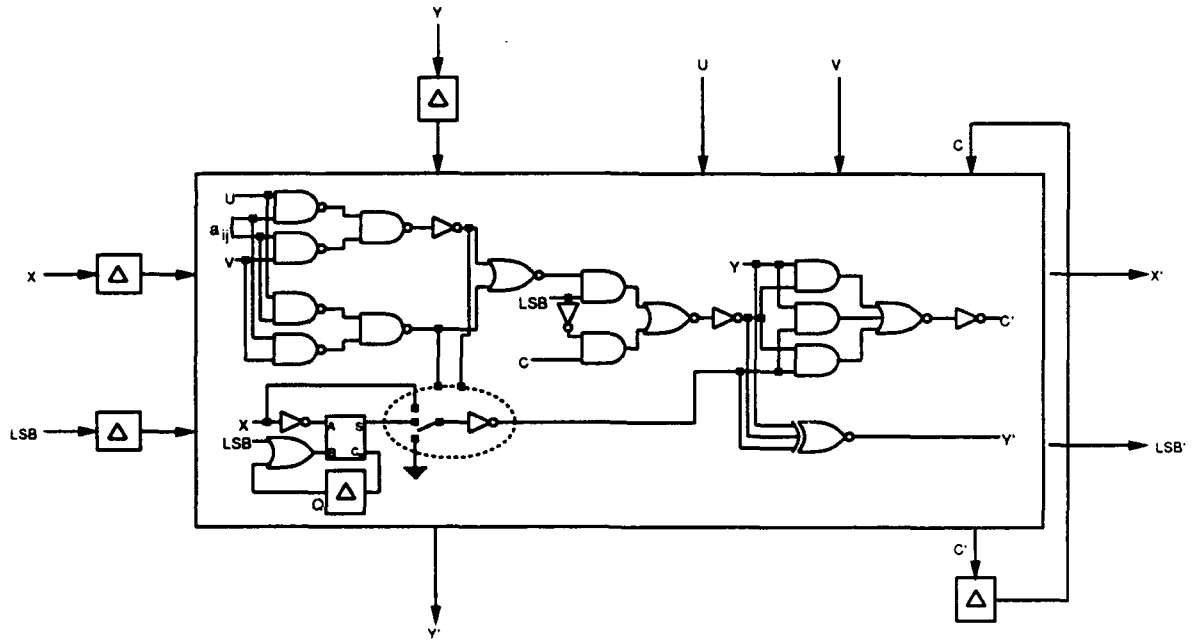


Figure 2: Logic diagram of one systolic array cell. The logic symbols are described in Appendix C.

resultant layout exhibits little regularity, as wires of various length connect the adders. Because the architecture requires routing between the adders, we refer to it as the *routed architecture.*

The adders can be organized in layers. The layers can then be stacked and interconnected using a "channel routing" software. Assume that the Winograd nesting method [2] is used for constructing an N-point algorithm from two smaller N_1-, and N_2-point algorithms. Let $N = N_1 N_2$, and N_1 and N_2 be relatively prime. The adders in the N-point implementation are now examined. Let L_1 and L_2 denote the number of layers of adders, and $\mathcal{A}_1$ and $\mathcal{A}_2$ denote the largest number of adders per layer, in the N_1-, and N_2-point implementations, respectively. Let M_1 denotes the number of multiplications in the N_1-point implementation. The number of layers (L) and the largest number of adders per layer ($\mathcal{A}$) in the N-point implementation are given by:

$$L = L_1 + L_2 \,, \tag{6}$$

$$\mathcal{A} = \max[N_2\mathcal{A}_1, M_1\mathcal{A}_2] \,. \tag{7}$$

The number of layers grows proportionally to $\log(N)$, whereas the number of adders per layer grows proportionally to N. The total number of adders is therefore proportional to $N\log(N)$. Note that this does not take into account the routing between

and through the layers, and the silicon area itself asymptotically grows at a higher rate[13]. Nevertheless, for the values of N considered here, the adders occupy more area than the interconnections, and their number is relevant.

In the routed architecture, the adders are divided into four groups. Two groups compute the real and imaginary additions before the multiplications; the two others compute the additions that follow the multiplications. Figure 3 shows a floorplan for the

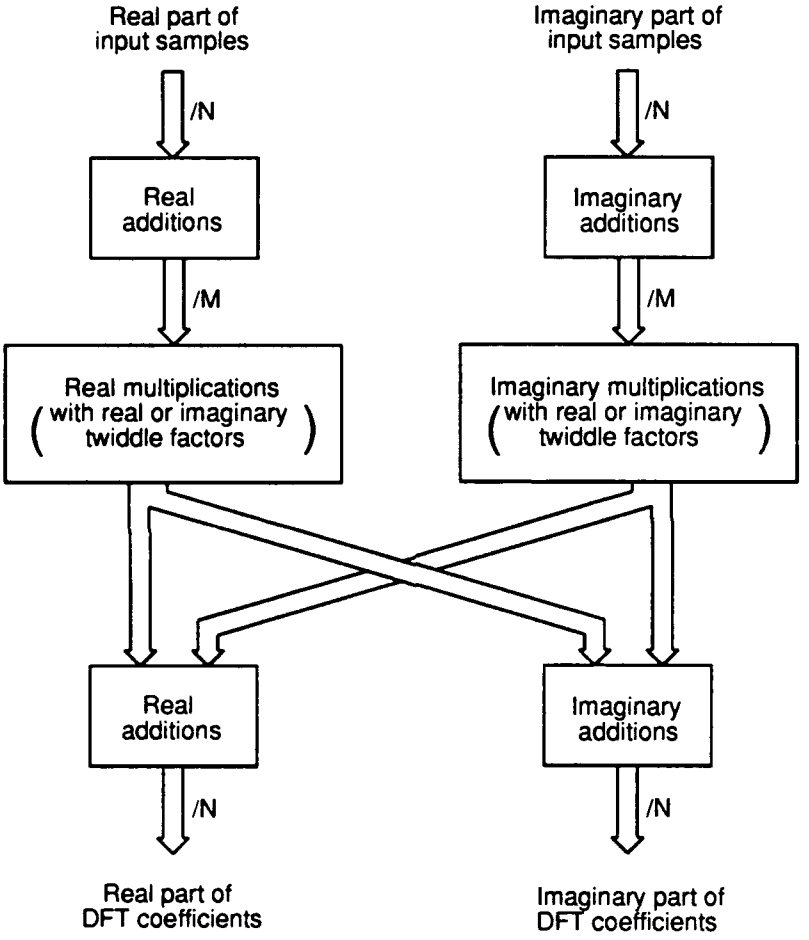


Figure 3: Floorplan of the routed architecture.

routed architecture, where the real parts of the input samples are processed on one side, and the imaginary parts on the other. The two sides are identical, except in the multipliers where some twiddle factors differ in sign. The computations of the two side can thus be carried out using separate circuits, provided that approximately $M/2$ pins are

[13]Thompson [42] has shown with an asymptotical analysis that the total area of any circuit computing the DFT in fixed time must grow proportionally to N^2.

available for data exchanges between the circuits and that the twiddle factors are programmable. In other words, two identical WFT circuits designed to compute DFTs of real-valued input sequences could therefore be connected together and compute DFTs of complex-valued sequences. This advantageous partitioning could not be implemented easily with the FFT.

In a bit-serial architecture, each addition can be implemented with a single cell of modest complexity. Each multiplication, on the other hand, requires a row of l_m cells, where l_m denotes the number of bits in the multiplicand (the twiddle factor). Therefore, for moderate transform sizes ($N < 100$), the multipliers generally occupy more silicon area than the four groups of adders. For higher values of N, the routing between the layers of adders occupies a higher percentage of the silicon area and may cause difficulties. It should be pointed out that the systolic architectures would also become impractical at that point.

The attainable throughput is equal to the clock rate divided by the number of bits per sample (l_s). In order to get a high clock rate (30MHz or higher), the "critical path" of the circuit, *i.e.* the electrical path with the longest propagation delay, must be minimized. In the routed architecture, the critical path may either be in the adders or in the multipliers. Indeed, the wires between the layers of adders may be of significant length and exhibit a large capacitance, having a significant effect on the circuit's speed. The adders should therefore be pipelined so that the data transfers between the different layers occur simultaneously. Inside each bit-serial multiplier, there is a carry chain where pipelining should also be applied. Pipelining shortens the critical path, but increases the latency of the circuit.

Figure 4 shows a block diagram of the routed architecture for Winograd's 5-point algorithm. Samples enter bit-serially at the top. They traverse the first two groups of adders (left and right), are multiplied by the twiddle factors, and traverse two more groups of adders. The bit-serial DFT coefficients then exit the circuit. Flip-flops have been inserted after every layer of adders, and at every three stages of multiplier cells, for pipelining.

An important issue when implementing the discrete Fourier transformation is the *partitioning*: can a computational load too large for a single device be distributed among several devices at a reasonable cost? A partitioning technique based on Winograd's nesting method is now proposed for the routed architecture.

Using Winograd's nesting method, an $N = N_1 N_2$-point algorithm can be constructed from two smaller N_1-, and N_2-point algorithms, conditional to N_1 and N_2 being relatively prime. Let A_1 and A_2 denote the number of additions, and M_1 and M_2 denote

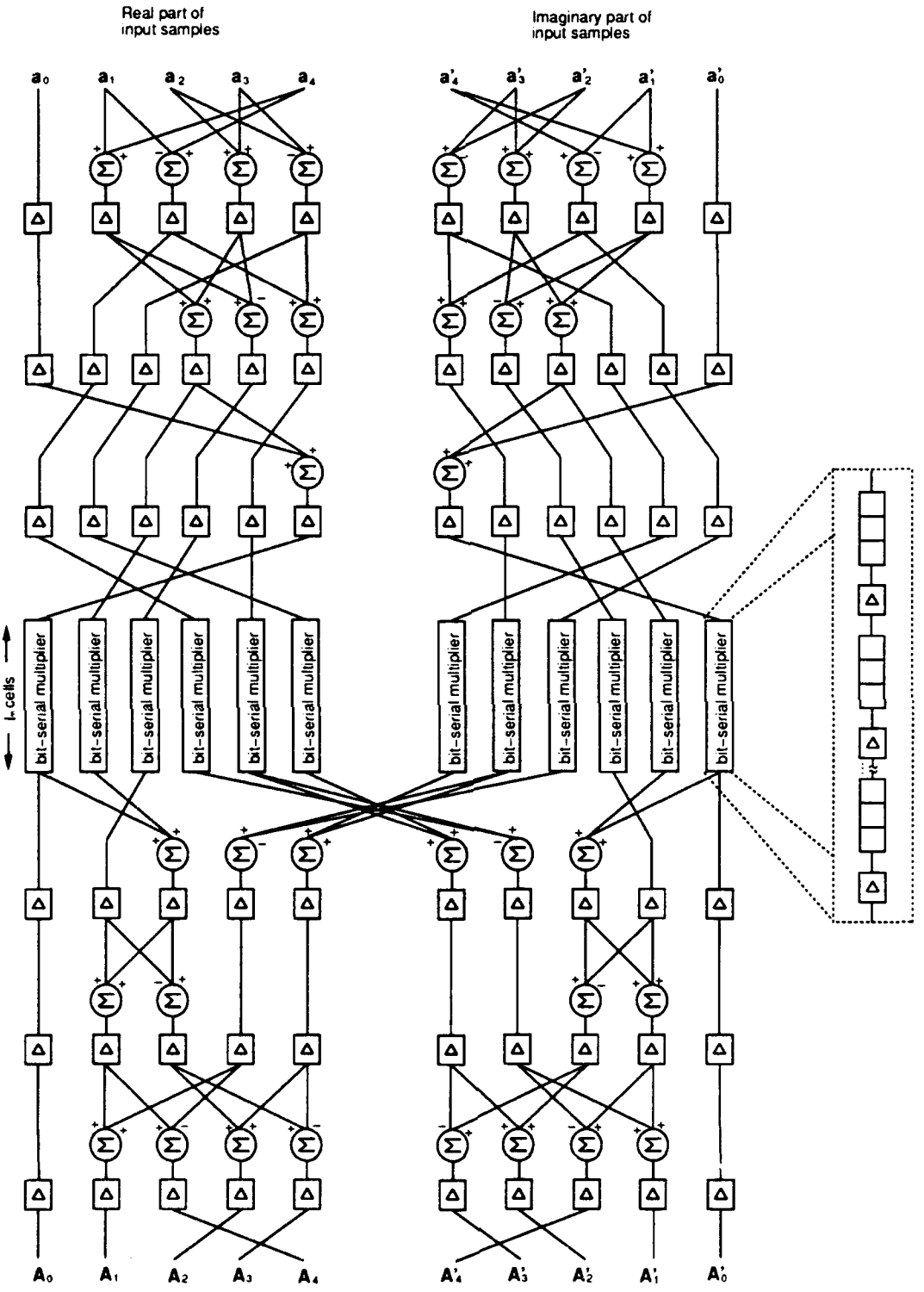


Figure 4: The routed architecture for Winograd's 5-point algorithm. In this particular case, the leftmost and rightmost multipliers could be replaced by flip-flops, since their twiddle factors are equal to one.

the number of multiplications, in the N_1-, and N_2-point algorithms, respectively. The constructed N-point algorithm then contains:

$$M = M_1 M_2 \quad \text{multiplications} \tag{8}$$

and

$$A = \min[N_2 A_1 + M_1 A_2,\ M_2 A_1 + N_1 A_2] \quad \text{additions.} \tag{9}$$

By examining the nesting method [2], the structure of the N-point algorithm appears as a "core" of M_1 N_2-point transformations "surrounded" by $A_1 N_2$ additions. The core is likely to be the expensive part because it contains all the multiplications. A natural way of partitioning it is between the N_2-point transformations. Thus a circuit computing one N_2-point transform and all the $A_1 N_2$ surrounding additions could compute N_2 of the N Fourier coefficients. Placing M_1 such circuits side by side would yield an N-point transformation machine. The appeal of this approach is twofold. First, the number of multipliers is minimal. Second, the circuits are identical[14]. The disadvantages are that the $A_1 N_2$ surrounding additions are duplicated M_1 times (once per circuit). Section 4.3 gives an example of the partitioning technique for $N_1 = 3$ and $N_2 = 20$. The example shows that modifications can be applied to the architecture for reducing its cost.

4.0 LOGIC DESIGN OF A 20-POINT WINOGRAD FOURIER TRANSFORMATION CIRCUIT

This section presents the design of a 20-point Winograd Fourier transformation circuit. The architecture chosen for implementing the circuit is the routed architecture presented in Section 3.0. We focus our attention on the design at the logic level, and give schematic diagrams of the cells required for building the circuit. The design has been given to a manufacturer for fabrication. Samples shall be available by the first quarter of 1992.

The Winograd Fourier transformation circuit, or *WFT circuit*, is designed to bit-serially accept and produce data in two's complement form. Surprisingly, bit-serial arithmetic components capable of accepting data in two's-complement notation are hard to find in the literature. Moreover, the few that we found generally turned out to be expensive in the number of gates. Most of the basic cells presented in this section are therefore either of our own, or the result of several modifications and iterations of a published design.

This section has six parts. First, the data format convention for communicating

[14]This is assuming that the twiddle factors can be programmed to suit the N-point transformation.

with the WFT circuit is described. The cells required for the additions are presented. Then the functionality required for a 60-point multi-circuit mode is included in the circuit. The multipliers are described in great detail. Next the position of the binary point at the output of the circuit is examined. The different modes of the circuit are explained with the associated control signals. Lastly, gate counts are given for all the cells and for the complete circuit.

4.1 DATA FORMAT

The data format has an impact on both the accuracy of the output and on the cost of the design. A fixed point format is cheaper to implement than a floating point. However, if the adders and multipliers use fixed point arithmetic, then overflows might occur in the circuit when the valid range determined by the number of bits l_s is exceeded. The following four strategies can be used to deal with this problem:

1. *Headroom provision*: The number of bits l_s is increased while keeping the sample values constant, so only a fraction of the input range available is used.

2. *Fixed scaling*: The outputs of some adders and multipliers are scaled down. The least significant bit (LSB) is dropped at given stages of the computations to increase the dynamic range available.

3. *Automatic scaling*: Scaling down is automatically applied where necessary. Thus overflows never occur. This is sometimes called "block floating point".

4. *Floating point*: Each individual data element is represented in a floating point format.

The first strategy (headroom provision) is the simplest, but it slows the bit-serial circuit down since l_s is increased. The second strategy (fixed scaling) is easily implemented and can reduce the probability of overflow. However, if the data is scaled down more than necessary, then precision will be lost at the output. Thus the second strategy is best combined with the first strategy: adding headroom allows delaying the scaling to latter stages, where the LSBs are non-significant anyways. The third strategy (automatic scaling) is intuitively appealing because just the minimum amount of scaling is applied. The last strategy (floating point) is the one that provides the most information at the output. It is more expensive than the three others.

The WFT circuit uses fixed scaling, which is fairly simple to implement. The block floating point and floating point strategies should be considered for future implementations.

Each complex sample has a real and an imaginary parts, whose values are represented in a fixed point, two's complement form. The 2^0 complex samples all enter the WFT circuit synchronously, in parallel, with their real and imaginary parts on separate input pins. The number of input pins is therefore equal to $2N = 40$. The least significant bits enter the circuit first, and the most significant bits last. Samples may be of any length l_s, but must be separated by one bit, or more, of *padding*. The values of the padding bits are discarded by the circuit. Along with the inputs samples, a control signal called "data valid in" (DVI) indicates whether the $2N$ accompanying data bits belong to samples (DVI = 1) or are simply padding bits (DVI = 0).

The 20 complex Fourier coefficients exit the WFT circuit in parallel, on a second set of $2N = 40$ pins, after a certain delay due to the pipelining in the circuit. The output format is similar to the input format, and a "data valid out" (DVO) signal generated by the circuit accompanies the Fourier coefficients. The Fourier coefficients are l_s bits long, with a binary point whose position depends on the input samples, the twiddle factors, and the amount of "scaling" that is being applied. They are separated by the same number of bits of padding as the corresponding input samples. However, the values of the padding bits may have changed and should be ignored. An "overflow" signal (OFO) is produced on the clock cycle following the delivery of the most significant bits of the Fourier coefficients. If this signal is high (OFO = 1), then an overflow has occurred somewhere in the circuit and the corresponding set of Fourier coefficients is invalid. The WFT circuit pursues its computations regardless of overflows. An overflow in one transformation does not affect the following transformations.

4.2 CELLS FOR ADDITION OPERATIONS

The 20-point WFTA[15] contains 108 complex (216 real) additions. These additions are divided into two sets: the additions before the multipliers (124 real additions), and the additions after the multipliers (92 real additions). Since the additions required for the real and imaginary data are the same, only one set of additions needs be considered. From this point, only the additions on real data are examined. It should be kept in mind that all the circuits shown in this section are duplicated in the WFT circuit.

The adders required for the "pre-multipliers" additions can be distributed on five layers. Figure 5 shows the 20 input samples $a_0, a_1, \ldots, a_{19}$, and the adders $s_1, s_2, \ldots, s_{62}$, organized in five horizontal layers. The layers are numbered from one to five, from top to bottom. The first layer contains the adders s_1 through s_{20} whose inputs are connected

[15]See Appendix A for the 20-point WFTA.

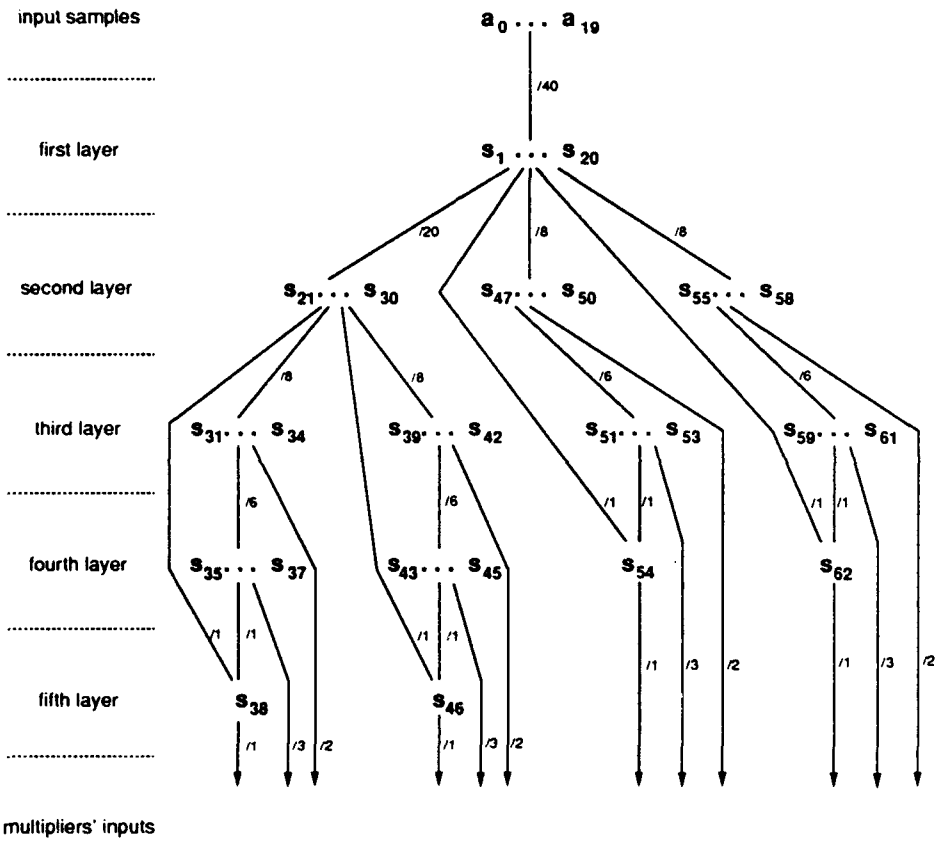


Figure 5: Pre-multipliers additions in a 20-point WFTA.

only to the input samples. The second layer contains the adders whose inputs connect to the first layer, the third layer contains adders whose inputs connect to the second layer, and so on. Each line in Fig. 5 may represent more than one data connection.

Sometimes it is imperative to pass data across one or more layers. For instance, on the second layer, two "through" wires allow the adders s_{54} and s_{62} on the fourth layer to get some input from the first layer. The data on a through wire is always transferred down as adders on one layer take their input only from the precedent layers. Counting the number of adders per layer, from top to bottom, yields 20, 18, 14, 8, and 2 adders, respectively. The 24 outputs of the adders enter into the multipliers.

The "post-multipliers" additions can be distributed on four layers. Figure 6 shows the adders and their data dependencies. From top to bottom, the layers have 14,

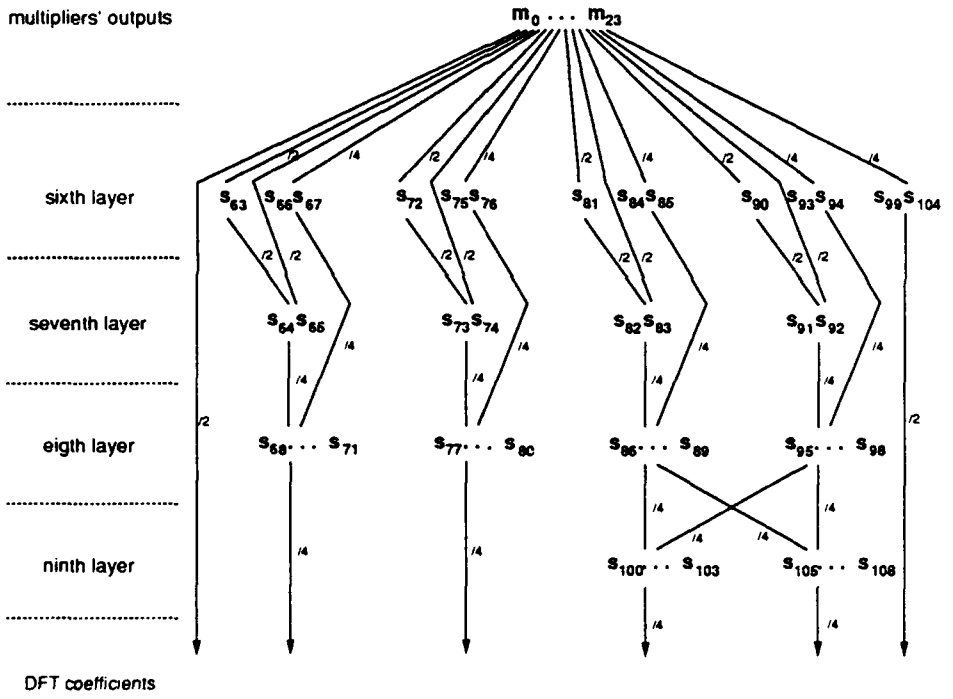


Figure 6: Post-multipliers additions in a 20-point WFTA.

8, 16, and 8 adders, respectively. The 20 outputs of the post-multipliers are the real parts of the Fourier coefficients A_0, A_1, ..., A_{19}.

As discussed in Section 3.0, pipelining the output of the adders is recommended for maintaining a high clock rate. Then the through wires must also be pipelined to ensure that the partial sums are synchronized.

In order to reduce the risk of overflow, programmable scalers are used for truncating the partial sums. If the scalers on a layer are "enabled", then the least significant

bit of every sum produced by that layer is discarded, making room for the most significant bit, and shifting the binary point by one position. Enabling or disabling the scalers thus allows tailoring the precision and dynamic range of the WFT circuit to the statistics of the input samples. Scalers must also be inserted along through wires.

Now that the global organization of the additions has been examined, the logic diagrams of the five associated cells are presented. There is a *padding cell*, an *adder*, an *overflow detection cell*, a *subtracter*, and a *hold-up cell*. Since the designs are mostly self-explanatory, the explanations are brief.

4.2.1 Padding Cell

The multipliers, in order to work properly, require that the most significant bit of the multiplicand be followed by at least one padding bit having the same value. This constraint can be met by making the samples go through *padding cells* upon their entry into the circuit. A padding cell is shown in Fig. 7. A sample enters the cell by the input X

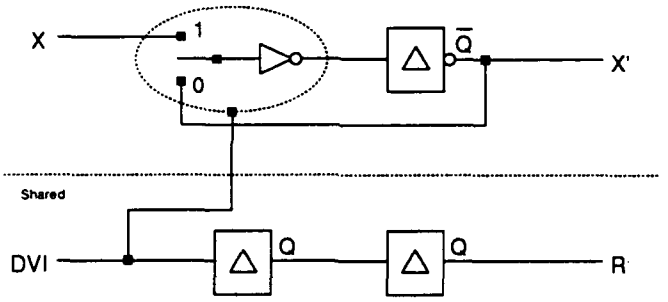


Figure 7: Logic diagram of a padding cell.

and exits by the output X'.

In the WFT circuit, the external DVI signal is delayed by one clock cycle with respect to the data and becomes a "reset" signal (R) that is used directly by the arithmetic cells. Before exiting the circuit, the data is delayed by one clock cycle with respect to the signal R, and the latter is output as DVO. The circuit shown under the padding cell in Fig. 7 transforms DVI into R. It is shared by all the padding cells of the circuit.

4.2.2 Adder

A bit-serial adder suitable for the WFT circuit is shown in Fig. 8. The two terms entering on X and Y are added together. The resultant sum exits on S. Scaling is implemented through a single multiplexer. This multiplexer is controlled by a circuit shared by all the adders on a same layer. The shared circuit also resets the carry between the additions.

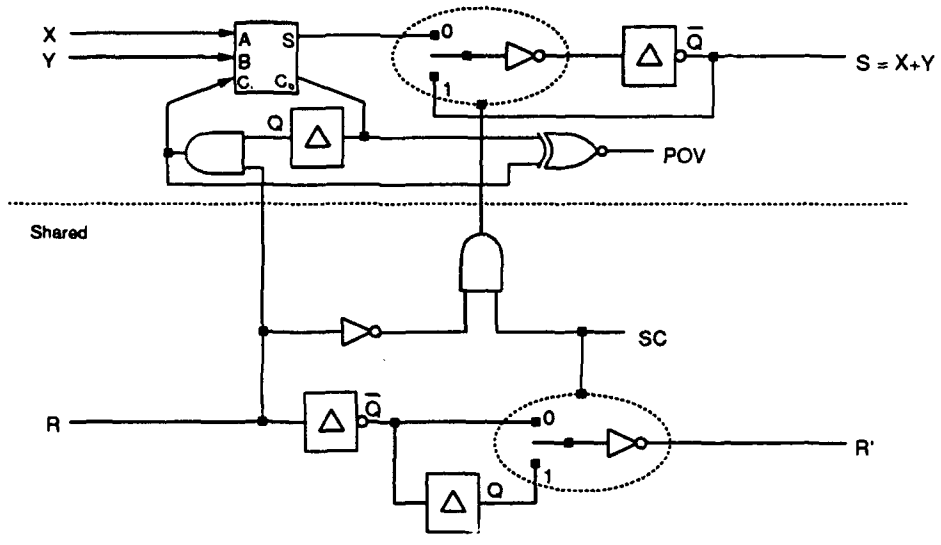


Figure 8: Logic diagram of a two's complement adder with scaling.

Setting the signal SC high (SC = 1) scales down all the outputs of the layer. A "partial overflow" signal (POV) is produced at every clock cycle by each adder.

4.2.3 Overflow Detection Cell

The signals POV produced by the adders of a layer are combined together in an *overflow detection cell*, as shown in Fig. 9. This cell declares whether an overflow has occurred (OVF = 1), or not (OVF = 0), in the layer.

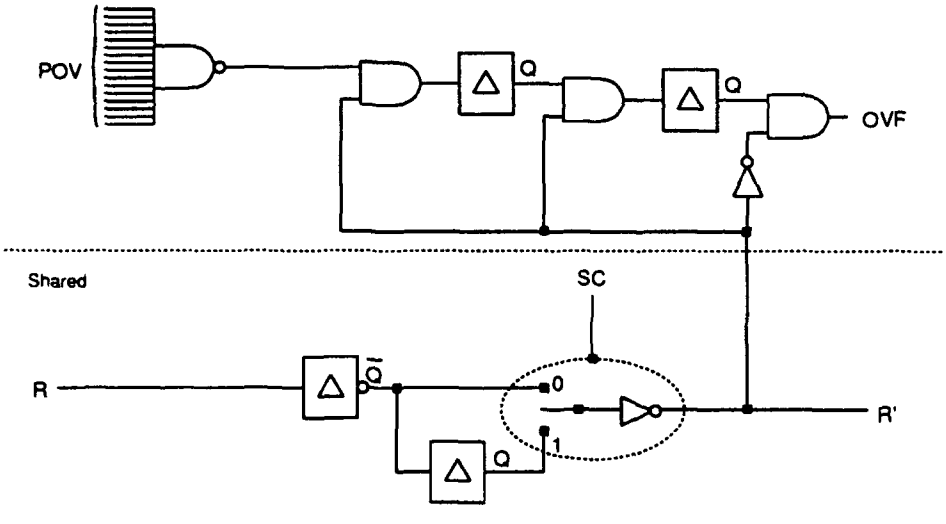


Figure 9: Logic diagram of an overflow detection cell.

Overflows may occur in several layers of the WFT circuit. The overflow signal of each layer must therefore be combined with the overflow signals of the other layers to yield the overall OFO signal. This is done using the scan circuitry shown in Fig. 10. The

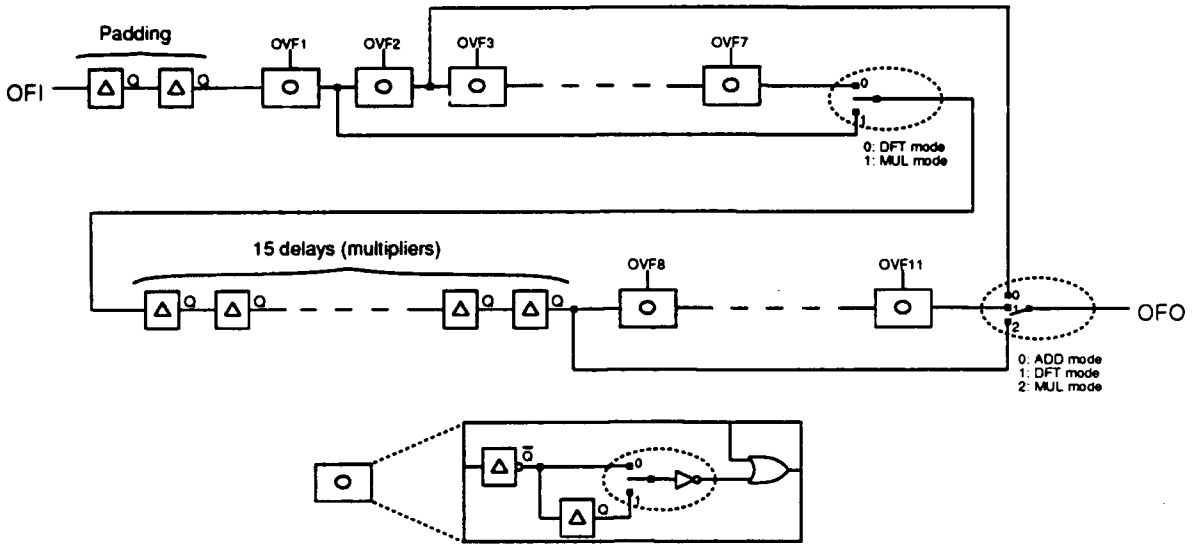


Figure 10: Logic diagram of the overall overflow circuitry.

OFO signal is output one clock cycle after the MSB of the Fourier coefficients. In a large system, the WFT circuit may be preceeded by other devices that may also overflow. An input to the overflow scan (OFI) has therefore be included in the design. Whenever a set of samples marked with an overflow enters the WFT circuit, the corresponding Fourier coefficients are thus automatically declared invalid.

4.2.4 Subtracter

A subtraction can be implemented either by fitting an adder with a sign inverter costing 18 gates, or by using a true subtracter. The second approach requires more design work, but yields a cell that has fewer gates. A subtracter cell is shown in Fig. 11. Remarkably, it has exactly the same number of gates as the adder of Fig. 8.

4.2.5 Hold-up cell

Pipelining must be applied evenly on all the width of a layer; otherwise the partial sums may loose their synchronism. This is also true of scaling; otherwise the position of the binary point may vary among the Fourier coefficients. Pipelining and scaling must therefore be applied to the data crossing a layer, or more, on through wires. A *hold-up cell*, such as the one shown in Fig. 12, serves that purpose.

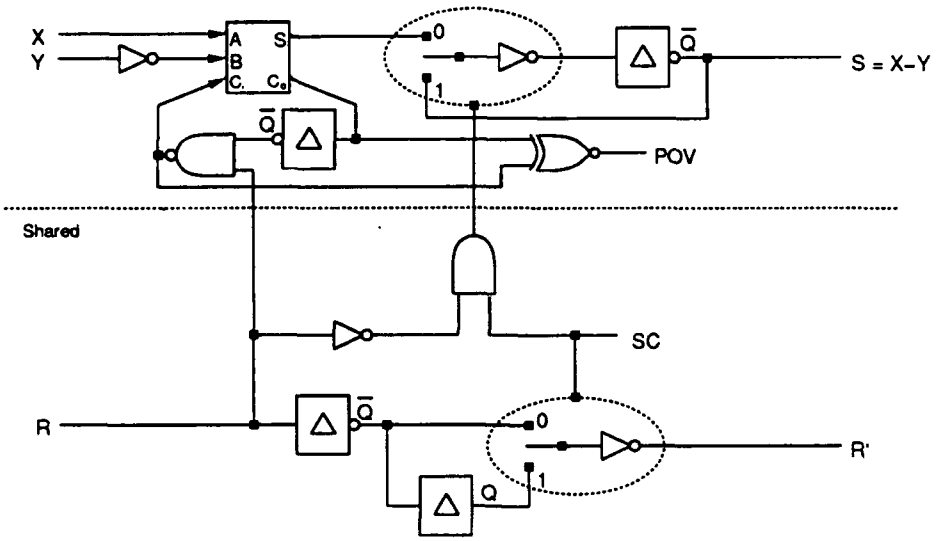


Figure 11: Logic diagram of a two's complement subtracter with scaling.

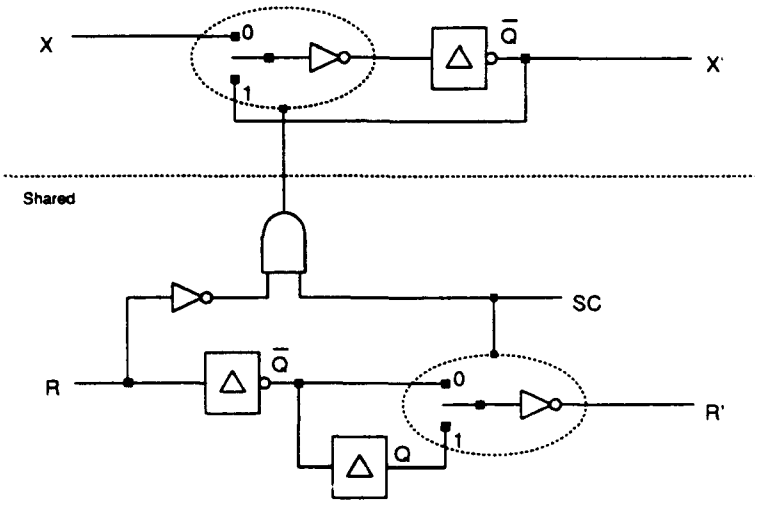


Figure 12: Logic diagram of a hold-up cell with scaling.

4.3 CELLS FOR A 60-POINT NESTED TRANSFORMATION

In Appendix A, a 60-point algorithm is derived from a 3-point algorithm and a 20-point algorithm. The resultant algorithm consists of a set of additions, three 20-point transformations, and another set of additions. The 20-point transformations are very similar to discrete Fourier transformations, at the exception of the twiddle factors that have different values. Assuming that the twiddle factors in the 20-point WFT circuit can be programmed to the required values, three circuits could therefore compute the "core" of the 60-point algorithm. Pursuing this idea, the extra adders required by the 60-point algorithm could be included in the WFT circuit. This would slightly increase the cost of the design, but greatly improve its versatility and usefulness.

There are many ways of dividing the computational load of the 60-point transformation among a set of identical devices. A three circuit configuration is probably the most efficient in terms of silicon area. However, the data flow between the circuits would require 240 data pins per circuit, and yield very high packaging costs. For many applications, the configuration shown in Fig. 13 with five circuits instead of three may provide a more balanced solution. This configuration requires only 160 data pins per circuit. Three of the circuits are used for additions and 20-point transforms: they accept the 60 complex samples arranged in three vectors $\mathbf{a_0}$, $\mathbf{a_1}$, and $\mathbf{a_2}$, and produce three intermediate vectors of results $\mathbf{M_0}$, $\mathbf{M_1}$, and $\mathbf{M_2}$. The 20-point transformations are denoted by $\mathbf{W_0}$, $\mathbf{W_1}$, and $\mathbf{W_2}$. Two circuits compute additions only: they accept $\mathbf{M_0}$, $\mathbf{M_1}$, and $\mathbf{M_2}$ and produce the Fourier coefficients in vectors $\mathbf{A_1}$, and $\mathbf{A_2}$. One "FIFO" circuit simply delays $\mathbf{M_0}$ to produce $\mathbf{A_0}$. It is not rigorously required, but has been included for convenience.

The Winograd nesting scheme, which has been used for building the 60-point WFTA, gets its indexing from the "Chinese Remainder Theorem." The order in which the input samples must be presented to the five circuits is therefore rather peculiar. The reader is referred to Appendix A for explanations on how the following vectors $\mathbf{a_0}$, $\mathbf{a_1}$, $\mathbf{a_2}$, $\mathbf{A_0}$, $\mathbf{A_1}$, and $\mathbf{A_2}$ are obtained:

$$\mathbf{a_0} = (a_0, a_{21}, a_{42}, a_3, a_{24}, a_{45}, a_6, a_{27}, a_{48}, a_9, a_{30}, a_{51}, a_{12}, a_{33}, a_{54}, a_{15}, a_{36}, a_{57}, a_{18}, a_{39}) .$$

$$\mathbf{a_1} = (a_{40}, a_1, a_{22}, a_{43}, a_4, a_{25}, a_{46}, a_7, a_{28}, a_{49}, a_{10}, a_{31}, a_{52}, a_{13}, a_{34}, a_{55}, a_{16}, a_7, a_{58}, a_{19}) .$$

$$\mathbf{a_2} = (a_{20}, a_{41}, a_2, a_{23}, a_{44}, a_5, a_{26}, a_{47}, a_8, a_{29}, a_{50}, a_{11}, a_{32}, a_{53}, a_{14}, a_{35}, a_{56}, a_{17}, a_{38}, a_{59}) .$$

$$\mathbf{A_0} = (A_0, A_{21}, A_{42}, A_3, A_{24}, A_{45}, A_6, A_{27}, A_{48}, A_9, A_{30}, A_{51}, A_{12}, A_{33}, A_{54}, A_{15}, A_{36}, A_{57}, A_{18}, A_{39}) .$$

$$\mathbf{A_1} = (A_{40}, A_1, A_{22}, A_{43}, A_4, A_{25}, A_{46}, A_7, A_{28}, A_{49}, A_{10}, A_{31}, A_{52}, A_{13}, A_{34}, A_{55}, A_{16}, A_7, A_{58}, A_{19}) .$$

$$\mathbf{A_2} = (A_{20}, A_{41}, A_2, A_{23}, A_{44}, A_5, A_{26}, A_{47}, A_8, A_{29}, A_{50}, A_{11}, A_{32}, A_{53}, A_{14}, A_{35}, A_{56}, A_{17}, A_{38}, A_{59}) .$$

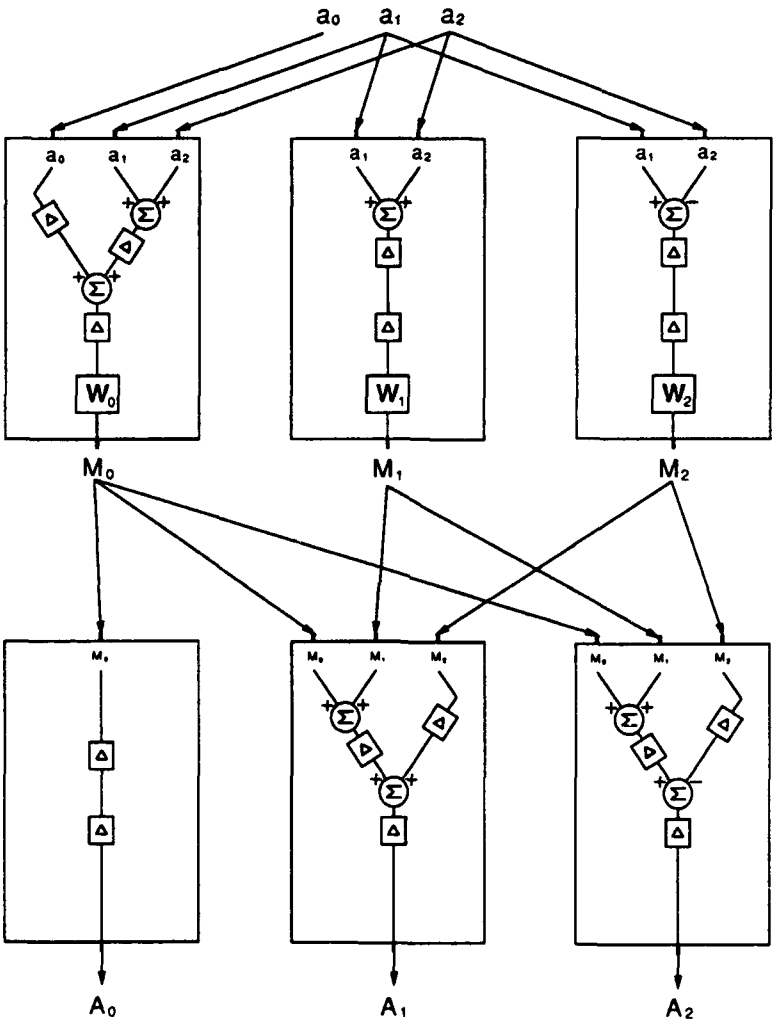


Figure 13: Block diagram of a 60-point DFT using five circuits. Note that the Fourier coefficients produced by the bottom left circuit run through a FIFO circuit just for synchronization with the other coefficients.

An economical way of implementing the extra additions that are required by the N-point WFTA consists of using programmable cells capable of either adding or subtracting. These cells can then be configured in agreement with the role of the circuit they belong to.

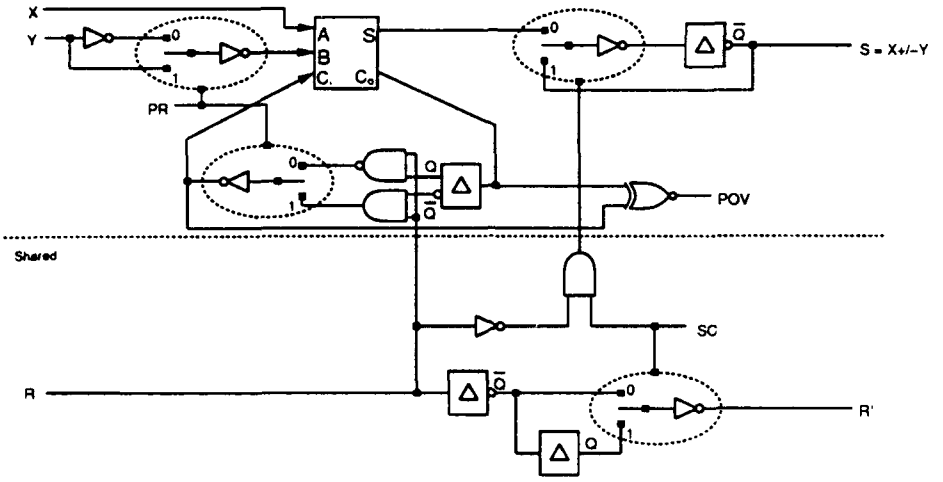


Figure 14: Logic diagram of a programmable cell capable of either adding or subtracting.

A cell that can be programmed for either adding or subtracting is shown in Fig. 14. If the control signal PR is set high (PR = 1), then the cell acts as a subtracter and produces X − Y; otherwise it produces X + Y. The usual timing and scaling control circuit is shown under the cell.

4.4 CELLS FOR MULTIPLICATION OPERATIONS

The representation of numbers in two's complement form, as is convenient for addition and subtraction, complicates the multiplication. Fortunately, the problem of implementing a bit-serial multiplier accepting two's complement data has been addressed by several authors. One solution consists of changing the numbers to a sign and magnitude notation, multiplying their magnitudes with a standard bit-serial multiplier, and transforming the result back into a correct two's complement number. The numbers can also be recoded with ternary digits to suit Booth's algorithm. However, a more elegant approach has been proposed by Lyon [43], who has succeeded in modifying the original pipeline multiplier developed by Jackson, Kaiser, and McDonald [44], which accepts positive data words only, allowing it to do correct two's complement multiplication. This last scheme is attractive for a number of reasons: it is modular, *i.e.* for a l_m-bit twiddle factor the multiplier consists of l_m identical cells; it rounds the products to the same length as the input data;

it computes the product at the same rate as the data is entered; and, lastly, it doesn't require data converters.

Lyon's fully two's complement multiplier can be modified to better suit the WFT circuit. First, it can be "re-timed" to reduce the amount of pipelining and the number of gates. Re-timing is a technique for shortening or lengthening the critical path, and thus the clock cycle duration, of VLSI circuits [45]. Then, the last two stages of the multiplier can be simplified.

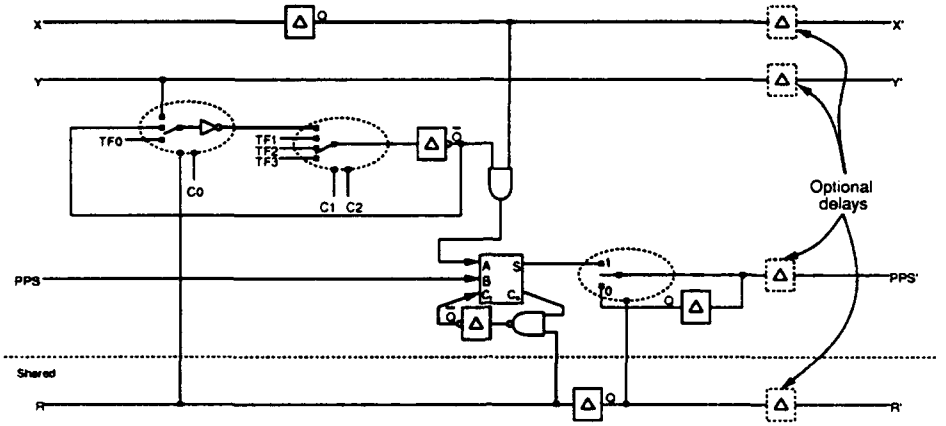
The two cells of the modified multiplier are shown in Fig. 15. In order to get a multiplier of length l_m, the first cell must be replicated $l_m - 1$ times, and this row must be terminated by the second cell. The l_s-bit data X travels through the l_m stages of the multiplier from left to right, one cell per clock cycle. The l_m-bit twiddle factor Y enters the multiplier simultaneously with the data, and each of its bits propagates to all the cells at once. The output Z of the last stage delivers the result, *i.e.* the l_s most significant bits of the product of X and Y. The data can be either shorter, equal in length, or longer than the twiddle factors. If it is shorter, then the twiddle factors must be stored before the multiplications and not changed.

The "partial product sum" input (PPS) of the first stage allows using an initial offset for rounding [43]. This "initial offset" (IO) is generated by the circuit that is shown in Fig. 16. Note that all the multipliers of the WFT circuit can share a single offset generation circuit.

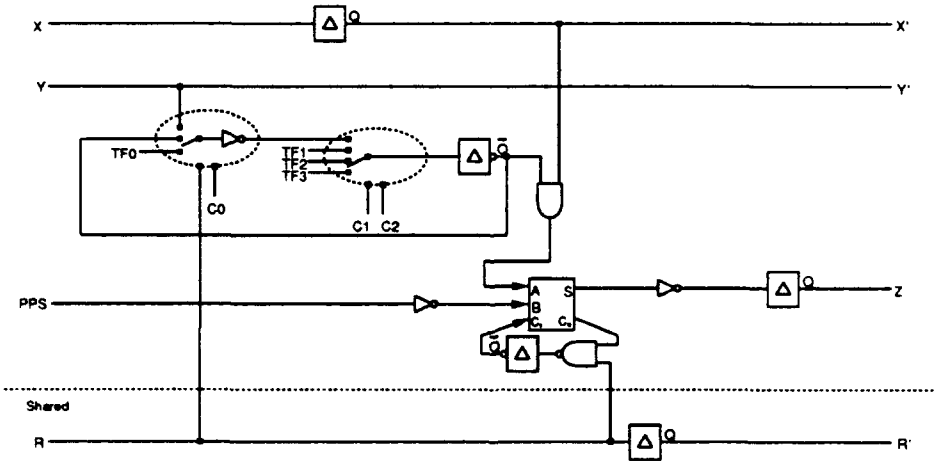
Each multiplier cell has two multiplexers for selecting one twiddle factor from a group of five possibilities. The multiplexers are controlled by the signals C0, C1, C2, and R. If C0 = 0, C1 = 0, and C2 = 0, then the multiplier reads its twiddle factor from Y. The four other possibilities (TF0, TF1, TF2, and TF3) correspond to the fixed values that are necessary for computing 20- and 60-point DFTs. Table 2 gives more detail on the multiplexers' control.

Appendix B provides all the twiddle factors that are required by the WFT circuit. Note that the multiplier shown in Fig. 15 inverts the bits of the twiddle factors TF1, TF2, and TF3. Hence the values given in appendix must be inverted before being stored in the multiplier. The bits of TF0 are not inverted and can be stored directly. The least significant bit must be stored into the first multiplier stage, where the multiplicands enter and the most significant bit into the last stage, where the product exits.

The four flip-flops marked "optional delays" in Fig. 15 allow pipelining the multipler to shorten the electrical path that otherwise runs from the PPS input of stage one to the Z output of stage l_m. Putting the flip-flops at every stage may be unnecessary, since the circuit's critical path would then surely move somewhere else, possibly in the



(a)



(b)

Figure 15: Logic diagram of a bit-serial multiplier cell. A *master* multiplier for l_m-bit twiddle factors can be built by juxtaposing $l_m - 1$ cells of type (a) with one cell of type (b).

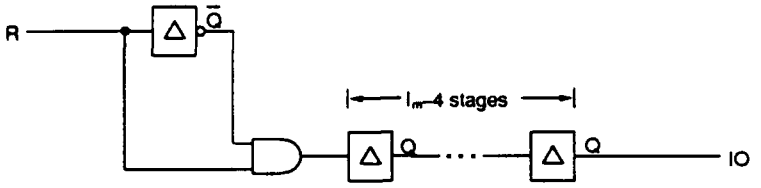


Figure 16: Logic diagram of an initial offset generation circuit for the bit-serial multiplier of Fig. 15. The signal IO is fed into the input PPS of the first multiplier's cell. The same circuit can be shared by all the multipliers.

Table 2: Control of the multipliers' multiplexers..

C0	C1	C2	R	Output	Note
0	0	0	0	Y	external twiddle factor
0	0	0	1	Q	stored twiddle factor
1	0	0	x	TF0	fixed value for 20 point DFT
x	0	1	x	TF1	fixed value for 60 point DFT (W_0)
x	1	0	x	TF2	fixed value for 60 point DFT (W_1)
x	1	1	x	TF3	fixed value for 60 point DFT (W_2)

routing between the adders, or in the output driving circuits. At first glance, inserting the flip-flops at every three stages should provide sufficient speed, and keep the cost of the multiplers low.

The 20-point WFT circuit contains 48 multipliers that are evenly divided into two groups. One group is fed with the real parts, and the other with the imaginary parts, of 24 complex intermediate results. The fixed twiddle factors are now examined further in detail, in an attempt to reduce the gate count of the multipliers. The values of the twiddle factors in the 24 multipliers that are fed with real numbers can be calculated using the equations:

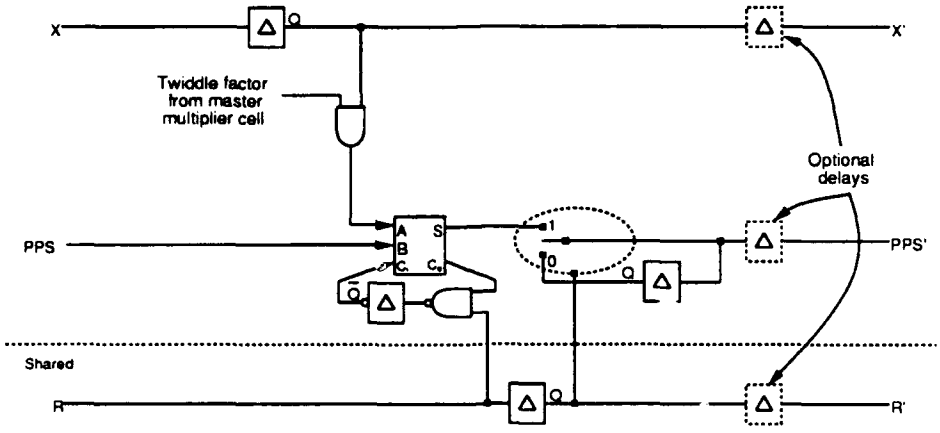
```
v = -pi/2
u = -8*pi/5

multiplier 0 : 1
multiplier 1 : ((cos(u)+cos(2*u))/2-1)
multiplier 2 : ((cos(u)-cos(2*u))/2)
multiplier 3 : (sin(u)+sin(2*u))
multiplier 4 : sin(2*u)
multiplier 5 : (sin(u)-sin(2*u))
```

```
multiplier 6 : 1
multiplier 7 : ((cos(u)+cos(2*u))/2-1)
multiplier 8 : ((cos(u)-cos(2*u))/2)
multiplier 9 : (sin(u)+sin(2*u))
multiplier 10 : sin(2*u)
multiplier 11 : (sin(u)-sin(2*u))
multiplier 12 : 1
multiplier 13 : ((cos(u)+cos(2*u))/2-1)
multiplier 14 : ((cos(u)-cos(2*u))/2)
multiplier 15 : (sin(u)+sin(2*u))
multiplier 16 : sin(2*u)
multiplier 17 : (sin(u)-sin(2*u))
multiplier 18 : sin(v)
multiplier 19 : sin(v)*((cos(u)+cos(2*u))/2-1)
multiplier 20 : sin(v)*((cos(u)-cos(2*u))/2)
multiplier 21 : sin(v)*(sin(u)+sin(2*u))
multiplier 22 : sin(v)*sin(2*u)
multiplier 23 : sin(v)*(sin(u)-sin(2*u))
```

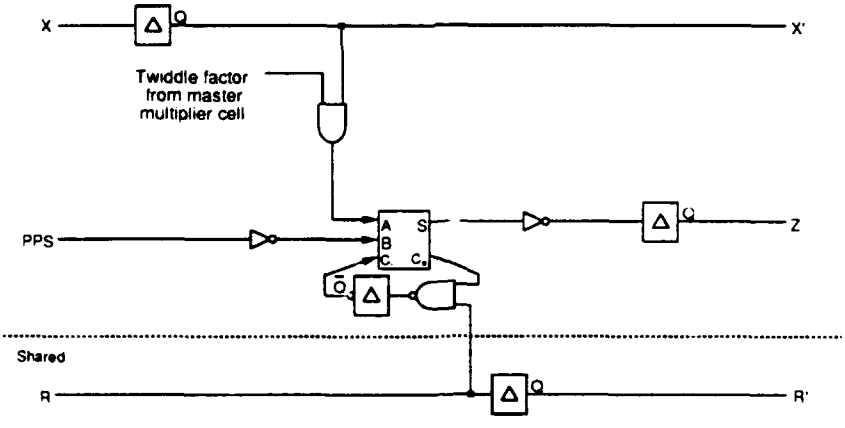
It is readily apparent that the twiddle factors of multipliers 0 through 5 are identical to those of multipliers 6 through 11 and 12 through 17. Since six of the 24 values appear three times each, these six values need only be stored once. One can therefore use just 12 multipliers with twiddle factor storage (*master* multipliers), and complete this set with 12 *slave* multipliers borrowing the twiddle factors of the first 12. The design of a slave multiplier is straightforward, and the cells obtained are shown in Fig. 17. Now considering the 24 multipliers fed with imaginary numbers, the situation is the same, the only difference being a sign inversion in the twiddle factors of multipliers 3, 4, 5, 9, 10, 11, 15, 16, 17, 18, 19, and 20[16].

In order to minimize the wiring length between the multipliers, every slave multiplier should be placed close to its master multiplier. The order 0, *6*, 12, *18*, 1, *7*, 13, *19*, 2, *8*, 14, *20*, 3, *9*, 15, *21*, 4, *10*, 16, *22*, 5, *11*, 17, and *23*, where the master multipliers are *italicized*, meets this constraint, and effectively places every master multiplier between its two slaves.

[16] This is correct for a 20-point transformation (TF0). For a 60-point transformation, this is also true for the sets TF1 and TF2; however, in TF3, it is the signs of multipliers 0, 1, 2, 6, 7, 8, 12, 13, 14, 21, 22, and 23 that are inverted.



(a)



(b)

Figure 17: Logic diagram of a bit-serial multiplier cell without twiddle factor control. A *slave* multiplier l_m-bit long is built by juxtaposing $l_m - 1$ cells of type (a) with one cell of type (b).

4.5 POSITION OF THE BINARY POINT

The position p_o of the binary point in the Fourier coefficients output by the WFT circuit is a function of its position in the input samples (p_i) and in the twiddle factors (p_m), and of the number of layers L_s whose scalers are active. Define the position of the binary point as the number of binary places between the point and the least significant bit. For example, the position of the binary point in "`101.01`" would be 2, whereas its position in "`11␣.`" would be -1. The position p_o of the binary point in the Fourier coefficients produced by the WFT circuit is given by:

$$p_o = p_i - L_s - (l_m - p_m - 1) \; . \tag{10}$$

4.6 OPERATING MODES AND CONTROL

In this section, the WFT circuit is examined globally. Its four operating modes, which are mutually exclusive, are described.

The first mode is a "discrete Fourier transformation" mode (*DFT*). This mode allows computing 20-point DFTs using a single circuit, as depicted in Fig. 18. The real

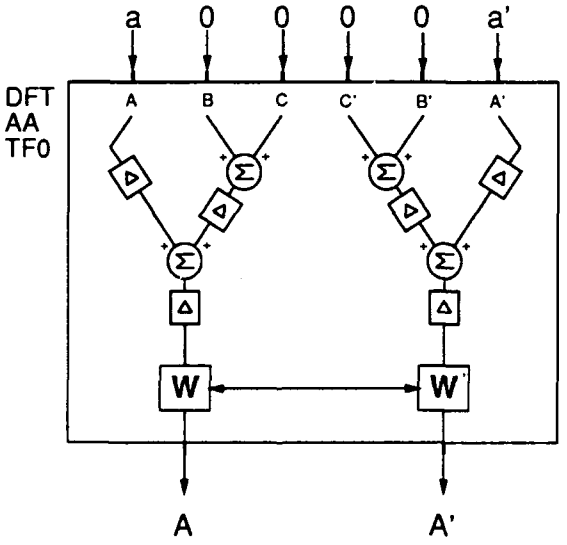


Figure 18: Block diagram of a WFT circuit computing 20-point transforms in its *DFT* mode .

and imaginary parts of the samples, which are respectively denoted by **a** and **a'**, enter the circuit at the top of the diagram and first traverse a tree of adders normally used in 60-point transformations. Since zeros are applied to the other inputs of the tree, the exiting samples are unchanged. The samples then enter into the transformation modules

denoted by **W** and **W′**. The real and imaginary parts of the Fourier coefficients, which are denoted by **A** and **A′**, exit at the bottom of the circuit.

The mode *DFT* is also used for computing the intermediate results $\mathbf{M}_0$, $\mathbf{M}_1$, and $\mathbf{M}_2$, in 60-point transformations. A five-cicuit configuration is shown in Fig. 19. In the 5-

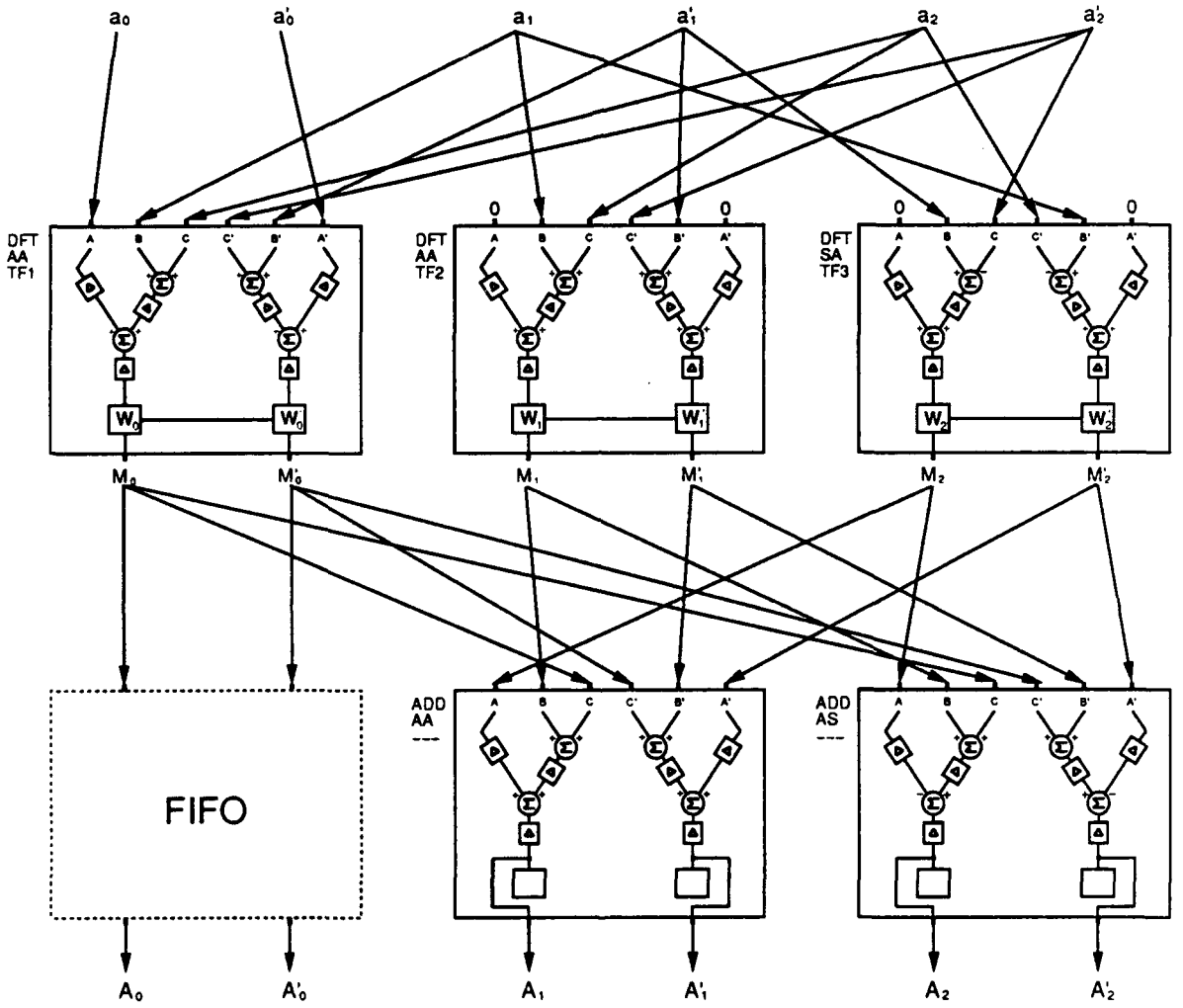


Figure 19: Block diagram of a 5-circuit array computing 60-point DFTs. The top three circuits are in the mode *DFT*. The bottom two are in the mode *ADD*. A sixth circuit (FIFO) simply delays 20 Fourier coefficients by two clock cycles.

circuit configuration, the intermediate sets of results $\mathbf{M}_0$, $\mathbf{M}_1$, and $\mathbf{M}_2$, enter two circuits that are in an "addition" mode (*ADD*), yielding 40 of the 60 Fourier coefficients. The remaining 20 Fourier coefficients are obtained by delaying the 20 intermediate results in the set $\mathbf{M}_0$ by two clock cycles.

The third mode of operation is a straightforward "multiplier" mode (*MUL*). A

circuit in this mode provides direct input and output to 24 of its multipliers. Figure 20 shows the corresponding internal data path. On one side of the circuit, the 12 inputs **x**

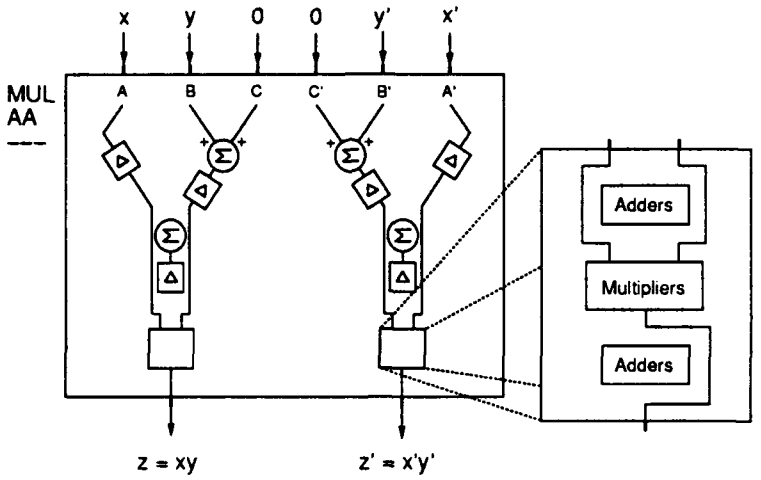


Figure 20: Block diagram of a WFT circuit in the mode *MUL*.

are multiplied by the 12 inputs **y** to yield the 12 products **z**. The same computations take place on the other side.

The last mode is a "test" mode (*TST*) for validation of the WFT circuit after fabrication. This mode is not a normal mode of operation. In this mode, all the flip-flops in the circuit become connected through a scan chain. Their values can be shifted out of the circuit and replaced by new values.

The programmable adders/subtracters required for 60-point transformations are distributed on two layers and controlled by a pair signals specifying whether they must "add and add" (*AA*), "add and subtract" (*AS*), "subtract and add" (*SA*), or "subtract and subtract" (*SS*). Figure 19 shows the mode of each circuit (*DFT* or *ADD*), the signals controlling the programmable adders/subtracters (*AA*, *AS*, or *SA*), and which twiddle factors are being used.

The scalers in the two layers of programmable adders/subtracters and in the nine layers of the 20-point transformation are controlled by a 3-bit signal that provides eight different settings. Table 3 shows the layers which are scaled down and which are not for each of the settings. A "1" in the table indicates that scaling is enabled, a "0" that it is disabled.

The internal data path of the WFT circuit is shown in Fig. 21. The picture, which provides wire counts, may appear complicated at first. However, the routing is straightforward and practical for current design tools and fabrication technology.

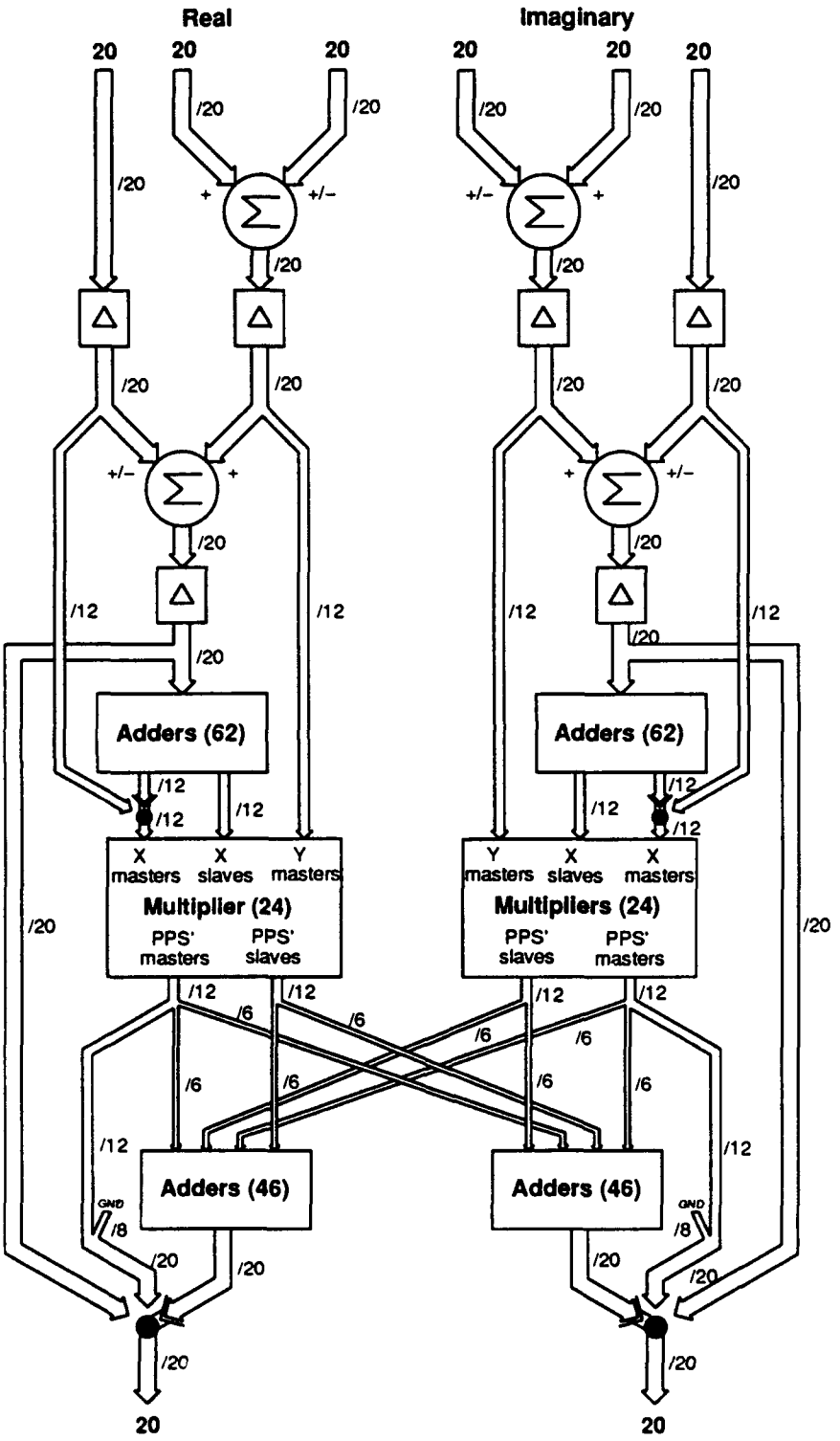


Figure 21: Internal data path of the WFT circuit.

Table 3: Scaling control..

Layer	Setting							
	0	1	2	3	4	5	6	7
Adders/subtracters, layer 1	0	0	0	0	0	0	0	1
Adders/subtracters, layer 2	0	0	0	0	0	0	1	1
20-point, pre-multipliers, layer 1	0	0	0	0	0	1	1	1
20-point, pre-multipliers, layer 2	0	0	0	0	1	1	1	1
20-point, pre-multipliers, layer 3	0	0	0	1	1	1	1	1
20-point, pre-multipliers, layer 4	0	0	1	1	1	1	1	1
20-point, pre-multipliers, layer 5	0	1	1	1	1	1	1	1
20-point, post-multipliers, layer 6	0	0	0	0	0	0	0	1
20-point, post-multipliers, layer 7	0	0	0	0	0	0	1	1
20-point, post-multipliers, layer 8	0	0	0	0	0	1	1	1
20-point, post-multipliers, layer 9	0	0	0	0	1	1	1	1

4.7 GATE COUNT

Using the logic diagrams of all the cells presented in this section, a gate count for the entire WFT circuit is now computed. The number of cells of each type is shown in Table 4.

By following the data progression in the circuit, one finds that there are 120 padding cells, one for each data input, and each cell has 13 gates. The data then traverses 80 adders/subtracters containing 44 gates each, and enter into the 20-point transformation. 120 adders and 96 subtracters, containing 36 gates each, are required for that transformation. Maintaining the synchronicity of the data path involves 204 hold-up cells with scaling: 40 are in the two layers of programmable adders/subtracters, and 164 are in the 20-point transformation modules. These cells are inexpensive at 13 gates each. A 55-gate overflow detection cell is required for each of the 11 layers of adders and subtracters. The master and slave multipliers require 576 multiplier cells having complexities ranging from 43 to 65 gates. They also require 360 hold-up cells containing 10 gates each. Then 40 hold-up cells delay the output of the data with respect to the DVO and OFO signals. For reconfiguring the data path to suit the various circuit modes, 40 2-to-1 and 24 3-to-1 multiplexers are required. After adding everything together, the WFT circuit ends up containing approximately 55 000 gates, and can therefore be implemented in a moderately large gate array.

Table 4: Cell and gate counts for the WFT circuit..

Cell Name	Count	Gates/Cell	Total Gates
padding cell	120	13	1 560
adder/subtracter	80	44	3 520
adder (with scaling)	120	36	4 320
subtracter (with scaling)	96	36	3 456
hold-up cell (with scaling)	204	13	2 652
overflow detection cell	11	≈55	605
offset generation	2	93	186
master mult. (stages 1–11)	264	65	17 160
master mult. (stage 12)	24	63	1 512
slave mult. (stages 1–11)	264	45	11 880
slave mult. (stage 12)	24	43	1 032
hold-up cell (without scaling)	400	10	4 000
multiplexers 2:1 (data path)	40	4	160
multiplexers 3:1 (data path)	24	5	120
Gates in circuit			52/,163

5.0 LOGIC SIMULATION

The WFT circuit has been simulated at the logic level for verifying the correctness and completeness of the design presented in Section 4.0 and measuring the effect of the truncation errors. The simulations have been carried out on small computers[17] using software written in the MATLAB programming language [46]. The circuit has been modeled at the gate level. Logic state transitions are synchronous, implying that propagation delays are not taken into account. The size of the source code is about 65 Kbytes. Simulation of the *DFT* mode takes about ten seconds per clock cycle. Computing the DFT of 20 15-bit complex samples takes 45 clock cycles from the time the circuit is reset until the most significant bit of the Fourier coefficients exits, and hence lasts for seven minutes and a half.

An example of simulation is now presented. The simulator is fed with 20 complex samples having real and imaginary 15-bit values picked up at random from the discrete interval $[-2\,048, 2\,047]$ with all values being equiprobable. As explained in Section 4.5, the binary point in the Fourier coefficients output by the circuit is shifted with respect to its position at the input. According to Equation (10), with $p_i = 0$, $L_s = 0$ (no scaling),

[17] Two workstations, a SUN SPARCStation I and a SUN IPC, have been used.

$l_m = 12$ and $p_m = 9$, the point is at position $p_o = -2$ of the output. The output of the WFT circuit must therefore by multiplied by four to yield the Fourier coefficients.

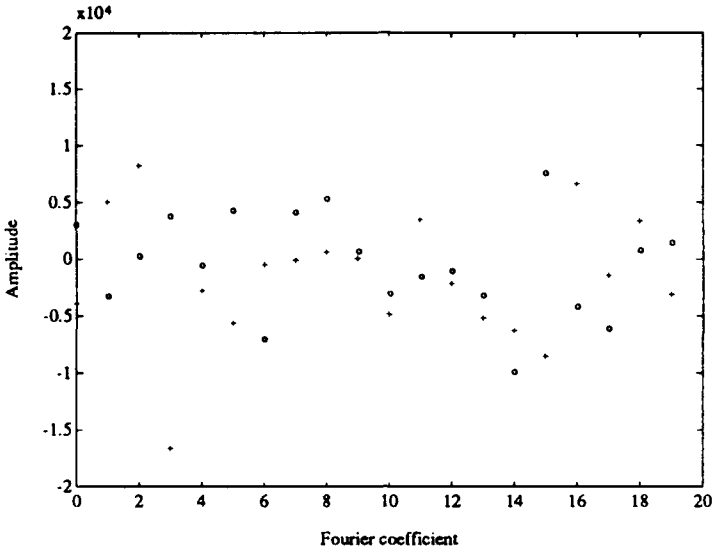


Figure 22: Plot of the Fourier coefficients computed by simulation of the WFT circuit for a set of input samples chosen at random in the interval $[-2048, 2047]$. Both the real ("o") and the imaginary ("+") parts of the coefficients are shown.

The Fourier coefficients obtained by simulation are shown in Fig. 22. Comparing the values obtained by simulation with the theoretical values yields the errors shown in Fig. 23. These errors result from the truncation errors of the twiddle factors stored in the multipliers. These initial truncation errors grow as they combine themselves in the adders and subtracters that follow. The Fourier coefficients of our example end up having only ten significant bits, whereas the input samples and the twiddle factors had twelve significant bits. The reader is referred to the literature for more detailed error analysis of Winograd's algorithms [47]-[52].

In the previous example, the input samples were assumed to be error-free, *i.e.* perfectly accurate. In practice, the input values themselves may be inaccurate, as a result of quantization, for example. This may further reduce the number of significant bits in the Fourier coefficients. Because the errors in the input samples combine themselves in the pre-multipliers additions, it would thus be a good idea to use more significant bits in the samples than in the twiddle factors, *i.e.* to set $l_s > l_m$.

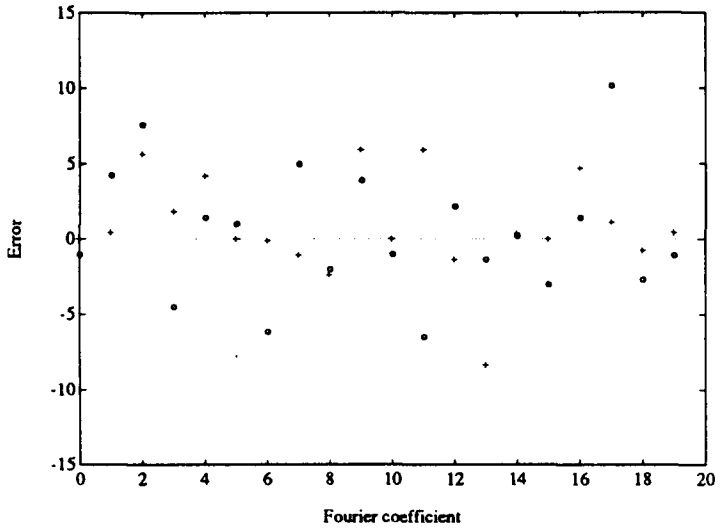


Figure 23: Plot of the differences, or errors, between the Fourier coefficients computed by simulation of the WFT circuit and their theoretical values (real "o" and imaginary "+" errors).

6.0 PRACTICAL CONSIDERATIONS

The testability, speed, and number of pins of the WFT circuit are examined in this section. These three issues are important in practice.

6.1 TESTABILITY

To verify if a circuit operates according to its specification, a comprehensive set of circuit stimuli with the expected outputs must be prepared. These are called *test vectors*. Test vectors can be categorized in two kinds.

The first kind of test vectors is intented to guaranty that a device under test has no fabrication defects, *i.e.* no ***faults***, and operates as predicted from its fabrication masks[18]. The fault coverage is generally measured by using a single "stuck at" fault model [53]. A score of 95% with this model is usually considered sufficient for prototyping. A 95% fault coverage means that one out of twenty defective devices having a single fault can pass the test, and end up in the customer's hands.

The generation and validation of the first kind of test vectors for the WFT circuit

[18]In practice, the generation and validation of test patterns is so difficult and costly that manufacturers often accept the risks of shipping insufficiently tested devices that may be defective.

has been assigned to the circuit manufacturer[19] which uses a generic testing approach applicable to any design. This structured approach requires special circuitry and increases the number of gates on the circuit by about 10 percent. The additional gates have been included in the gate counts of Section 4.7.

The second kind of test vectors are aimed at verifying that the manufactured devices, and the manufacturer's circuit model, does what the designer expects. These *functional* test vectors are meant to ensure that the circuit has no *design flaws*. They can be used in gate-level circuit simulations, *i.e.* where propagation delays are taken into account, and during fabrication, normally as a complement to the first kind of test patterns.

Functional test vectors cannot be exhaustive because with modern circuit densities the number of possible input combinations is too large. Therefore, the circuit designer must prepare ad hoc functional test vectors aimed at detecting common design flaws.

It would be tedious to describe all the functional test vectors that have been prepared for the WFT circuit. A quick overview of the various categories of test vectors should be sufficient to illustrate the concept of functional testability. The eight categories of test vectors are listed below, along with short descriptions:

1. *Test adder*: Compute maximum positive value plus zero, one, and minus one. Compute minimum negative value plus zero, one, and minus one. Compute maximum value plus minimum value.

2. *Test subtracter*: Same test as for the adder.

3. *Test programmable adder/subtracter*: Try the control signals (*AA*, *AS*, *SA*, and *SS*) while in the mode *ADD*.

4. *Test overflow*: In the mode *DFT*, with scaling enabled and disabled, trigger overflows in an adder using two positive and two negative numbers, and do the same in a subtracter with two numbers with different signs. Repeat for pre-, and post-multipliers layers as well as for real and imaginary sides of circuit. Try to overflow several adders per layer, and several layers at once.

5. *Test twiddle factors*: In the mode *DFT*, apply input vectors to circuit such that every bit of every twiddle factor can be observed at the output. Repeat for the four different settings: TF0, TF1, TF2, and TF3.

[19] The manufacturer is LSI Logic Co. of Canada.

6. *Test multiplier*: For 15-bit long data, separated by 1- and 10-bit long padding, multiply very small and very large values, testing all the four possible sign combinations. Make sure that at least one observable result would be different if the initial offset circuitry didn't work.

7. *Test 20-point DFT*: Input random complex data with mixed signs and verify Fourier coefficients produced by circuit. Try a sinewave that produces an overflow and one that does not. Repeat with scaling.

8. *Test 60-point DFT*: Apply to the circuit the inputs it would get if it were used five times in succession to compute 60-point discrete Fourier transforms.

Generating the functional test vectors using the logic simulator took several weeks of CPU time on our computers. The functional test is by no means exhaustive, but it should provide enough evidence for judging whether the circuits produced by the manufacturer are functional or not.

6.2 SPEED

The speed of a WFT circuit manufactured in a 0.7μm CMOS gate array technology is now discussed. Preliminary investigation of the circuit has indicated that its clock rate will be limited by a signal path in the multipliers. Assuming that pipelining flip-flops are inserted at every three stages of multiplier cells[20], the critical path would run through three full adders and three multiplexers. Electrical simulations indicate a maximum clock rate of 30 MHz. This is an approximation, since statistical estimates of layout-dependent parameters were used.

Assuming that the samples are $l_s = 15$ bits long, with one bit of padding between the samples of two successive transforms, a 30 MHz WFT circuit could compute a transform in $16 \times 0.033\text{ns} = 0.53\mu\text{s}$, *i.e.* compute over 1.8 million transforms per second. This corresponds to throughputs of 37 million and 111 million samples per second for 20-point and 60-point transformations, respectively.

6.3 PIN COUNT

The pin requirement of the WFT circuit is given in Table 5. It turns out that 196 pins are necessary, of which 164 are used for transferring data and indicating overflows, 10 for control, 2 for testability, 1 for the external clock and 19 for power. No tri-state pads are

[20]See Section 4.1 for an explanation of these flip-flops.

Table 5: Input/output requirement of the WFT circuit..

Name	Description	Pins	in/out
Data in	six groups of 20	120	in
Data out	two groups of 20	40	out
Mode	mode: *DFT*, *ADD*, *MUL*, or *TST*	2	in
Adder/Subtr. control	*AA*, *AS*, *SA*, or *SS*	2	in
Twiddle Factors control	TF0, TF1, TF2, or TF3	2	in
Scaling control	eight different settings	3	in
Circuit reset	RST	1	in
Data Valid In	DVI	1	in
Data Valid Out	DVO	1	out
Overflow In	OFI	1	in
Overflow Out	OFO	1	out
Scan In	for testability	1	in
Scan Out	for testability	1	out
Clock	CLK	1	in
Power	VDD, GND	19	n.a.
Total		196	

being used: all pins are either for input (134), output (43) or power (19). Packages with 196 pins are currently available.

7.0 COST COMPARISON

In this section, the routed architecture is compared to the systolic architecture of Ward *et al* [27] and to a parallel FFT architecture from a cost standpoint. The unit measure of cost is the logic gate, for lack of a better unit that would take into consideration the routing areas. The routed WFTA and the parallel FFT architectures require routing, whereas the systolic architecture does not.

The number of additions and multiplications in the WFTA can be derived from Equations (8) and (9). The additions and multiplications in the FFT can be calculated by [19]

$$A_{FFT} = (N/2)(-10 + 7\log_2 N) + 8\,. \tag{11}$$

and

$$M_{FFT} = (N/2)(-10 + 3\log_2 N) + 8\,. \tag{12}$$

The number of arithmetic operations being known, the gate count of each architecture

can be computed. Assume that each multiplier has 12 stages, and that each stage consists of one multiplier cell containing 72 gates. Adders and subtracters cost 36 gates each. The arrays in the systolic architecture contain $336NM_W$ gates[21]. Denote by G_{FFT}, G_{routed}, and $G_{systolic}$, the numbers of gates in the FFT, routed, and systolic architectures, respectively. These gate counts can be calculated by the equations:

$$G_{FFT} = (12 \cdot 72 \cdot M_{FFT}) + (36 \cdot A_{FFT}) , \quad (13)$$

$$G_{routed} = (12 \cdot 72 \cdot M_W) + (36 \cdot A_W) , \quad (14)$$

$$G_{systolic} = (12 \cdot 72 \cdot M_W) + (336 \cdot N \cdot M_W) . \quad (15)$$

The costs of the three architectures are shown in Fig. 24 as a function of the number of points N. Of the three, the routed architecture appears to be the least expensive.

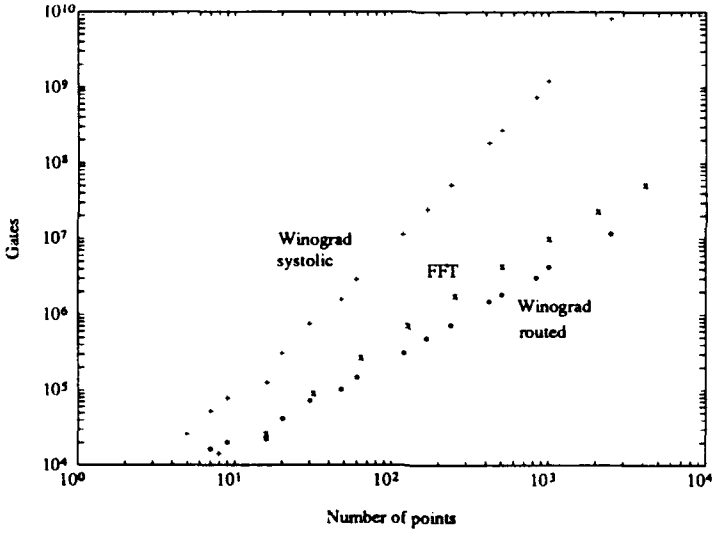


Figure 24: Plot of the the number of gates in the FFT, routed, and systolic architectures, as a function of the number of points N.

The FFT architecture contains two to three times more gates than the routed WFTA for comparable transform sizes. The systolic architecture is much more expensive than the two others.

[21] See Section 2.0 for an analysis of the systolic architecture cost.

8.0 CONCLUSION

In this report, a routed architecture for the Winograd Fourier transform algorithms (WFTA) has been presented. This bit-serial architecture maps an N-point WFTA directly onto a VLSI circuit. The resultant layout exhibits little regularity among the adders, but it covers a small area and can be generated by computer-aided design tools. The nesting method invented by Winograd has been proposed as a means of partitioning a large transformation into several pieces implemented on individual circuits. One advantage of this partitioning approach is that it minimizes the number of multipliers, which are very expensive. Another advantage is that the netted circuits can be all of the same type. This reduces the design time and the number of mask sets. The main disadvantage is that it requires more input/output pins than some other approaches.

The logic design of a 20-point Winograd fourier transformation (WFT) circuit has been presented in detail. Data formats, which have an impact on the output accuracy and design cost, have been examined. Floorplans for the pre- and post-multipliers additions have been proposed, along with logic diagrams for the adding and subtracting cells. Overflow detection has been included in the design. Low cost multipliers for two's complement input and output data have been designed. The circuit can be programmed to compute either 20-point DFTs by itself, or 60-point DFTs when it is connected to four other circuits. The circuit contains about 55 000 gates and has 196 pins. The whole design has been simulated on a computer to verify the correctness of its logic and measure the accuracy of the output data. A comprehensive set of test vectors has been designed for verifying the functionality of the circuit samples.

Overall, the routed architecture appears to be attractive for computing moderate size DFTs at very high speeds. The routed architecture can also be combined with partitioning techniques like the prime factor algorithm for computing larger DFTs. The possible applications include electronic warfare, image, radar, speech, and sonar processing.

APPENDIX

A.0 DERIVATION OF WINOGRAD FOURIER TRANSFORM ALGORITHMS FOR 20 POINTS AND 60 POINTS

In this section we derive Winograd Fourier transformation algorithms (WFTA) for 20 and 60 points. We assume that the reader has a basic understanding of the nesting method of Winograd [2].

A.1 DERIVATION OF A 20-POINT ALGORITHM

Suppose that one wants to derive an algorithm for computing N-point DFTs. Let $N = N_1 N_2$, where N_1 and N_2 are relatively prime. Assume that an algorithm is known for computing N_1-point DFTs using A_1 additions and M_1 multiplications, and that another algorithm is known for computing N_2-point DFTs using A_2 additions and M_2 multiplications. Then the N_1-point and N_2-point algorithms can be "nested" to yield an N-point algorithm requiring $M_1 M_2$ multiplications and $N_2 A_1 + M_1 A_2$ additions.

A 20-point algorithm is now derived from the 4- and 5-point algorithms. Let $N = 20$, $N_1 = 4$, and $N_2 = 5$. The resultant 20-point algorithm contains

$$M = M_1 M_2 = 4 \cdot 6 = 24 \quad \text{complex multiplications,}$$

and

$$A = N_2 A_1 + M_1 A_2 = 5 \cdot 8 + 4 \cdot 17 = 108 \quad \text{complex additions.}$$

With $N_1 = 5$ and $N_2 = 4$, it would contain instead

$$A' = N_2 A_1 + M_1 A_2 = 4 \cdot 17 + 6 \cdot 8 = 116 \quad \text{complex additions}$$

and be more expensive.

By the Chinese Remainder Theorem (CRT), every integer $0 \leq n \leq N - 1$ can be represented by the pair (n_1, n_2) such that $n_1 = n \bmod N_1$ and $n_2 = n \bmod N_2$. Taking $20 = 4 \cdot 5$, we get the mapping:

0 - (0,0)	1 - (1,1)	2 - (2,2)	3 - (3,3)
4 - (0,4)	5 - (1,0)	6 - (2,1)	7 - (3,2)
8 - (0,3)	9 - (1,4)	10 - (2,0)	11 - (3,1)
12 - (0,2)	13 - (1,3)	14 - (2,4)	15 - (3,0)
16 - (0,1)	17 - (1,2)	18 - (2,3)	19 - (3,4)

APPENDIX

which put into lexicographical order yields: 0, 16, 12, 8, 4, 5, 1, 17, 13, 9, 10, 6, 2, 18, 14, 15, 11, 7, 3, and 19.

Let us define

$$\mathbf{a}_0 = \begin{pmatrix} a_0 \\ a_{16} \\ a_{12} \\ a_8 \\ a_4 \end{pmatrix}, \quad \mathbf{a}_1 = \begin{pmatrix} a_5 \\ a_1 \\ a_{17} \\ a_{13} \\ a_9 \end{pmatrix}, \quad \mathbf{a}_2 = \begin{pmatrix} a_{10} \\ a_6 \\ a_2 \\ a_{18} \\ a_{14} \end{pmatrix}, \quad \mathbf{a}_3 = \begin{pmatrix} a_{15} \\ a_{11} \\ a_7 \\ a_3 \\ a_{19} \end{pmatrix},$$

$$\mathbf{A}_0 = \begin{pmatrix} A_0 \\ A_{16} \\ A_{12} \\ A_8 \\ A_4 \end{pmatrix}, \quad \mathbf{A}_1 = \begin{pmatrix} A_5 \\ A_1 \\ A_{17} \\ A_{13} \\ A_9 \end{pmatrix}, \quad \mathbf{A}_2 = \begin{pmatrix} A_{10} \\ A_6 \\ A_2 \\ A_{18} \\ A_{14} \end{pmatrix}, \quad \mathbf{A}_3 = \begin{pmatrix} A_{15} \\ A_{11} \\ A_7 \\ A_3 \\ A_{19} \end{pmatrix},$$

and apply the following 4-point algorithm [2] to these vectors:

$$\begin{aligned} \mathbf{s}_1 &= \mathbf{a}_0 + \mathbf{a}_2 & \mathbf{s}_2 &= \mathbf{a}_0 - \mathbf{a}_2 \\ \mathbf{s}_3 &= \mathbf{a}_1 + \mathbf{a}_3 & \mathbf{s}_4 &= \mathbf{a}_1 - \mathbf{a}_3 \\ \mathbf{s}_5 &= \mathbf{s}_1 + \mathbf{s}_3 & \mathbf{s}_6 &= \mathbf{s}_1 - \mathbf{s}_3 \end{aligned}$$

$$\begin{aligned} \mathbf{M}_1 &= \mathbf{W} \cdot \mathbf{s}_5 & \mathbf{M}_2 &= \mathbf{W} \cdot \mathbf{s}_6 \\ \mathbf{M}_3 &= \mathbf{W} \cdot \mathbf{s}_2 & \mathbf{M}_4 &= i \sin v \, \mathbf{W} \cdot \mathbf{s}_4 \end{aligned}$$

$$\mathbf{s}_7 = \mathbf{M}_3 + \mathbf{M}_4 \qquad \mathbf{s}_8 = \mathbf{M}_3 - \mathbf{M}_4$$

$$\begin{aligned} \mathbf{A}_0 &= \mathbf{M}_1 & \mathbf{A}_1 &= \mathbf{s}_7 \\ \mathbf{A}_2 &= \mathbf{M}_2 & \mathbf{A}_3 &= \mathbf{s}_8 \end{aligned}$$

where $\mathbf{W}$ denotes the 5-point transformation given in Section 2.2, and $v = -\frac{\pi}{2}$. A 20-point Winograd Fourier transformation algorithm is obtained.

The remaining step consists of computing the values of u and v which appear in the twiddle factors of the algorithm (u comes from the 5-point, v from the 4-point).

APPENDIX

Beginning with u, the value of b must be calculated using the equation [2]:

$$(e^{-\frac{2\pi i}{N}})^{(0,1)} = (e^{-\frac{2\pi i}{N_2}})^{\mathsf{b}} . \tag{A.1}$$

The CRT indicates that $(0,1)$ corresponds to 16, thus

$$(e^{-\frac{2\pi i}{20}})^{16} = (e^{-\frac{2\pi i}{5}})^{\mathsf{b}} \tag{A.2}$$

and

$$\mathsf{b} = 4 . \tag{A.3}$$

The value of u in the N-point algorithm is equal to b times its value in the N_2-point algorithm $(-\frac{2\pi}{5})$:

$$u = 4(-\frac{2\pi}{5}) = -\frac{8\pi}{5} . \tag{A.4}$$

To obtain v, the value of a must be computed using the equation [2]:

$$(e^{-\frac{2\pi i}{N}})^{(1,0)} = (e^{-\frac{2\pi i}{N_1}})^{\mathsf{a}} . \tag{A.5}$$

The CRT maps $(1,0)$ into 5, thus

$$(e^{-\frac{2\pi i}{20}})^{5} = (e^{-\frac{2\pi i}{4}})^{\mathsf{a}} \tag{A.6}$$

and

$$\mathsf{a} = 1 . \tag{A.7}$$

Consequently, the value of v is unchanged:

$$v = -\frac{\pi}{2} . \tag{A.8}$$

This completes the derivation of the 20-point WFTA. The resultant algorithm is now simply stated in the same raw format that was input to our simulator for validation. The additions and multiplications are all on complex data.

APPENDIX

A.2 20-POINT ALGORITHM

```
v = -pi/2
u = -8*pi/5

s1 = a0+a10    s6 = a0-a10    s11 = a5+a15
s2 = a16+a6    s7 = a16-a6    s12 = a1+a11
s3 = a12+a2    s8 = a12-a2    s13 = a17+a7
s4 = a8+a18    s9 = a8-a18    s14 = a13+a3
s5 = a4+a14    s10 = a4-a14    s15 = a9+a19

s16 = a5-a15    s21 = s1+s11    s26 = s1-s11
s17 = a1-a11    s22 = s2+s12    s27 = s2-s12
s18 = a17-a7    s23 = s3+s13    s28 = s3-s13
s19 = a13-a3    s24 = s4+s14    s29 = s4-s14
s20 = a9-a19    s25 = s5+s15    s30 = s5-s15

s31 = s22+s25    s32 = s22-s25    s33 = s24+s23
s34 = s24-s23    s35 = s31+s33    s36 = s31-s33
s37 = s32+s34    s38 = s35+s21
s39 = s27+s30    s40 = s27-s30    s41 = s29+s28
s42 = s29-s28    s43 = s39+s41    s44 = s39-s41
s45 = s40+s42    s46 = s43+s26
s47 = s7+s10    s48 = s7-s10    s49 = s9+s8
s50 = s9-s8    s51 = s47+s49    s52 = s47-s49
s53 = s48+s50    s54 = s51+s6
s55 = s17+s20    s56 = s17-s20    s57 = s19+s18
s58 = s19-s18    s59 = s55+s57    s60 = s55-s57
s61 = s56+s58    s62 = s59+s16

m0 = s38
m1 = ((cos(u)+cos(2*u))/2-1)*s35
m2 = ((cos(u)-cos(2*u))/2)*s36
m3 = j*(sin(u)+sin(2*u))*s32
m4 = j*sin(2*u)*s37
m5 = j*(sin(u)-sin(2*u))*s34
m6 = s46
m7 = ((cos(u)+cos(2*u))/2-1)*s43
m8 = ((cos(u)-cos(2*u))/2)*s44
```

APPENDIX

```
m9 = j*(sin(u)+sin(2*u))*s40
m10 = j*sin(2*u)*s45
m11 = j*(sin(u)-sin(2*u))*s42
m12 = s54
m13 = ((cos(u)+cos(2*u))/2-1)*s51
m14 = ((cos(u)-cos(2*u))/2)*s52
m15 = j*(sin(u)+sin(2*u))*s48
m16 = j*sin(2*u)*s53
m17 = j*(sin(u)-sin(2*u))*s50
m18 = j*sin(v)*s62
m19 = j*sin(v)*((cos(u)+cos(2*u))/2-1)*s59
m20 = j*sin(v)*((cos(u)-cos(2*u))/2)*s60
m21 = j*sin(v)*j*(sin(u)+sin(2*u))*s56
m22 = j*sin(v)*j*sin(2*u)*s61
m23 = j*sin(v)*j*(sin(u)-sin(2*u))*s58

s63 = m0+m1   s64 = s63+m2   s65 = s63-m2
s66 = m3-m4   s67 = m4+m5   s68 = s64+s66
s69 = s64-s66   s70 = s65+s67   s71 = s65-s67
s72 = m6+m7   s73 = s72+m8   s74 = s72-m8
s75 = m9-m10   s76 = m10+m11   s77 = s73+s75
s78 = s73-s75   s79 = s74+s76   s80 = s74-s76
s81 = m12+m13   s82 = s81+m14   s83 = s81-m14
s84 = m15-m16   s85 = m16+m17   s86 = s82+s84
s87 = s82-s84   s88 = s83+s85   s89 = s83-s85
s90 = m18+m19   s91 = s90+m20   s92 = s90-m20
s93 = m21-m22   s94 = m22+m23   s95 = s91+s93
s96 = s91-s93   s97 = s92+s94   s98 = s92-s94

s99 = m12+m18   s104 = m12-m18
s100 = s86+s95   s105 = s86-s95
s101 = s88+s97   s106 = s88-s97
s102 = s89+s98   s107 = s89-s98
s103 = s87+s96   s108 = s87-s96

A0 = m0   A5 = s99   A10 = m6   A15 = s104
A16 = s68   A1 = s100   A6 = s77   A11 = s105
A12 = s70   A17 = s101   A2 = s79   A7 = s106
A8 = s71   A13 = s102   A18 = s80   A3 = s107
A4 = s69   A9 = s103   A14 = s78   A19 = s108
```

APPENDIX

A.3 DERIVATION OF A 60-POINT ALGORITHM

With the help of the 20-point algorithm, we now derive an algorithm for $60 = 3 \cdot 20$ points which contains $M = 3 \cdot 24 = 72$ complex multiplications and $A = 20 \cdot 6 + 3 \cdot 108 = 444$ complex additions. The procedure is exactly the same as for the 20-point algorithm. Note that the factors 3, 4, and 5, are mutually prime. If they were not, the Winograd nesting method could not be used.

Using the Chinese Remainder Theorem, the inputs and outputs are reordered as follows: 0, 21, 42, 3, 24, 45, 6, 27, 48, 9, 30, 51, 12, 33, 54, 15, 36, 57, 18, 39, 40, 1, 22, 43, 4, 25, 46, 7, 28, 49, 10, 31, 52, 13, 34, 55, 16, 37, 58, 19, 20, 41, 2, 23, 44, 5, 26, 47, 8, 29, 50, 11, 32, 53, 14, 35, 56, 17, 38, 59. Let us define the vectors

$$\begin{aligned}
\mathbf{a}_0^t &= (a_0\ a_{21}\ a_{42}\ a_3\ a_{24}\ a_{45}\ a_6\ a_{27}\ a_{48}\ a_9\ a_{30}\ a_{51}\ a_{12}\ a_{33}\ a_{54}\ a_{15}\ a_{36}\ a_{57}\ a_{18}\ a_{39}) \\
\mathbf{a}_1^t &= (a_{40}\ a_1\ a_{22}\ a_{43}\ a_4\ a_{25}\ a_{46}\ a_7\ a_{28}\ a_{49}\ a_{10}\ a_{31}\ a_{52}\ a_{13}\ a_{34}\ a_{55}\ a_{16}\ a_{37}\ a_{58}\ a_{19}) \\
\mathbf{a}_2^t &= (a_{20}\ a_{41}\ a_2\ a_{23}\ a_{44}\ a_5\ a_{26}\ a_{47}\ a_8\ a_{29}\ a_{50}\ a_{11}\ a_{32}\ a_{53}\ a_{14}\ a_{35}\ a_{56}\ a_{17}\ a_{38}\ a_{59}) \\
\mathbf{A}_0^t &= (A_0\ A_{21}\ A_{42}\ A_3\ A_{24}\ A_{45}\ A_6\ A_{27}\ A_{48}\ A_9\ A_{30}\ A_{51}\ A_{12}\ A_{33}\ A_{54}\ A_{15}\ A_{36}\ A_{57}\ A_{18}\ A_{39}) \\
\mathbf{A}_1^t &= (A_{40}\ A_1\ A_{22}\ A_{43}\ A_4\ A_{25}\ A_{46}\ A_7\ A_{28}\ A_{49}\ A_{10}\ A_{31}\ A_{52}\ A_{13}\ A_{34}\ A_{55}\ A_{16}\ A_{37}\ A_{58}\ A_{19}) \\
\mathbf{A}_2^t &= (A_{20}\ A_{41}\ A_2\ A_{23}\ A_{44}\ A_5\ A_{26}\ A_{47}\ A_8\ A_{29}\ A_{50}\ A_{11}\ A_{32}\ A_{53}\ A_{14}\ A_{35}\ A_{56}\ A_{17}\ A_{38}\ A_{59})
\end{aligned}$$

and apply to these vectors the 3-point algorithm [2]

$$\mathbf{s}_1 = \mathbf{a}_1 + \mathbf{a}_2 \qquad \mathbf{s}_2 = \mathbf{a}_1 - \mathbf{a}_2 \qquad \mathbf{s}_3 = \mathbf{s}_1 + \mathbf{a}_0$$

$$\mathbf{M}_0 = \mathbf{W} \cdot \mathbf{s}_3 \qquad \mathbf{M}_1 = (\cos w - 1)\mathbf{W} \cdot \mathbf{s}_1 \qquad \mathbf{M}_2 = i \sin w\, \mathbf{W} \cdot \mathbf{s}_2$$

$$\mathbf{s}_4 = \mathbf{M}_0 + \mathbf{M}_1 \qquad \mathbf{s}_5 = \mathbf{s}_4 + \mathbf{M}_2 \qquad \mathbf{s}_6 = \mathbf{s}_4 - \mathbf{M}_2$$

$$\mathbf{A}_0 = \mathbf{M}_0 \qquad \mathbf{A}_1 = \mathbf{s}_5 \qquad \mathbf{A}_2 = \mathbf{s}_6$$

where this time $\mathbf{W}$ denotes a 20-point transformation and $w = -\frac{2\pi}{3}$. A 60-point Winograd Fourier transformation algorithm is obtained.

The new values of u, v, and w are easily found. Since $(0, 1)$ corresponds to 21, $\mathbf{b}$ is equal to 7, and hence

$$u = 7(-\frac{8\pi}{5}) = -\frac{56\pi}{5}\ ,$$

and

$$v = 7(-\frac{\pi}{2}) = -\frac{7\pi}{2}\ .$$

Similarly, $(1,0)$ corresponds to 40, a is equal to 2, and the value of w in the 60-point algorithm is:

$$w = 2(-\frac{2\pi}{3}) = -\frac{4\pi}{3} .$$

The resultant 60-point Winograd Fourier transform algorithm is given bellow. Again all the additions and multiplications are on complex data.

A.4 60-POINT ALGORITHM

```
u = -(56/5)*pi
v = -(7/2)*pi
w = -(4/3)*pi

s0 = a40+a20    s20 = a40-a20    s40 = s0+a0
s1 = a1+a41     s21 = a1-a41     s41 = s1+a21
s2 = a22+a2     s22 = a22-a2     s42 = s2+a42
s3 = a43+a23    s23 = a43-a23    s43 = s3+a3
s4 = a4+a44     s24 = a4-a44     s44 = s4+a24
s5 = a25+a5     s25 = a25-a5     s45 = s5+a45
s6 = a46+a26    s26 = a46-a26    s46 = s6+a6
s7 = a7+a47     s27 = a7-a47     s47 = s7+a27
s8 = a28+a8     s28 = a28-a8     s48 = s8+a48
s9 = a49+a29    s29 = a49-a29    s49 = s9+a9
s10 = a10+a50   s30 = a10-a50    s50 = s10+a30
s11 = a31+a11   s31 = a31-a11    s51 = s11+a51
s12 = a52+a32   s32 = a52-a32    s52 = s12+a12
s13 = a13+a53   s33 = a13-a53    s53 = s13+a33
s14 = a34+a14   s34 = a34-a14    s54 = s14+a54
s15 = a55+a35   s35 = a55-a35    s55 = s15+a15
s16 = a16+a56   s36 = a16-a56    s56 = s16+a36
s17 = a37+a17   s37 = a37-a17    s57 = s17+a57
s18 = a58+a38   s38 = a58-a38    s58 = s18+a18
s19 = a19+a59   s39 = a19-a59    s59 = s19+a39

s61 = s40+s50   s66 = s40-s50    s71 = s45+s55
s62 = s56+s46   s67 = s56-s46    s72 = s41+s51
s63 = s52+s42   s68 = s52-s42    s73 = s57+s47
s64 = s48+s58   s69 = s48-s58    s74 = s53+s43
s65 = s44+s54   s70 = s44-s54    s75 = s49+s59
```

APPENDIX

```
s76 = s45-s55   s81 = s61+s71   s86 = s61-s71
s77 = s41-s51   s82 = s62+s72   s87 = s62-s72
s78 = s57-s47   s83 = s63+s73   s88 = s63-s73
s79 = s53-s43   s84 = s64+s74   s89 = s64-s74
s80 = s49-s59   s85 = s65+s75   s90 = s65-s75

s91 = s82+s85   s92 = s82-s85   s93 = s84+s83
s94 = s84-s83   s95 = s91+s93   s96 = s91-s93
s97 = s92+s94   s98 = s95+s81
s99 = s87+s90   s100 = s87-s90   s101 = s89+s88
s102 = s89-s88   s103 = s99+s101   s104 = s99-s101
s105 = s100+s102   s106 = s103+s86
s107 = s67+s70   s108 = s67-s70   s109 = s69+s68
s110 = s69-s68   s111 = s107+s109   s112 = s107-s109
s113 = s108+s110   s114 = s111+s66
s115 = s77+s80   s116 = s77-s80   s117 = s79+s78
s118 = s79-s78   s119 = s115+s117   s120 = s115-s117
s121 = s116+s118   s122 = s119+s76

m0 = s98
m1 = ((cos(u)+cos(2*u))/2-1)*s95
m2 = ((cos(u)-cos(2*u))/2)*s96
m3 = j*(sin(u)+sin(2*u))*s92
m4 = j*sin(2*u)*s97
m5 = j*(sin(u)-sin(2*u))*s94
m6 = s106
m7 = ((cos(u)+cos(2*u))/2-1)*s103
m8 = ((cos(u)-cos(2*u))/2)*s104
m9 = j*(sin(u)+sin(2*u))*s100
m10 = j*sin(2*u)*s105
m11 = j*(sin(u)-sin(2*u))*s102
m12 = s114
m13 = ((cos(u)+cos(2*u))/2-1)*s111
m14 = ((cos(u)-cos(2*u))/2)*s112
m15 = j*(sin(u)+sin(2*u))*s108
m16 = j*sin(2*u)*s113
m17 = j*(sin(u)-sin(2*u))*s110
m18 = j*sin(v)*s122
m19 = j*sin(v)*((cos(u)+cos(2*u))/2-1)*s119
```

APPENDIX

```
m20 = j*sin(v)*((cos(u)-cos(2*u))/2)*s120
m21 = j*sin(v)*j*(sin(u)+sin(2*u))*s116
m22 = j*sin(v)*j*sin(2*u)*s121
m23 = j*sin(v)*j*(sin(u)-sin(2*u))*s118

s123 = m0+m1   s124 = s123+m2   s125 = s123-m2
s126 = m3-m4   s127 = m4+m5   s128 = s124+s126
s129 = s124-s126   s130 = s125+s127   s131 = s125-s127
s132 = m6+m7   s133 = s132+m8   s134 = s132-m8
s135 = m9-m10   s136 = m10+m11   s137 = s133+s135
s138 = s133-s135   s139 = s134+s136   s140 = s134-s136
s141 = m12+m13   s142 = s141+m14   s143 = s141-m14
s144 = m15-m16   s145 = m16+m17   s146 = s142+s144
s147 = s142-s144   s148 = s143+s145   s149 = s143-s145
s150 = m18+m19   s151 = s150+m20   s152 = s150-m20
s153 = m21-m22   s154 = m22+m23   s155 = s151+s153
s156 = s151-s153   s157 = s152+s154   s158 = s152-s154

s159 = m12+m18   s164 = m12-m18
s160 = s146+s155   s165 = s146-s155
s161 = s148+s157   s166 = s148-s157
s162 = s149+s158   s167 = s149-s158
s163 = s147+s156   s168 = s147-s156

s169 = s0+s10   s174 = s0-s10   s179 = s5+s15
s170 = s16+s6   s175 = s16-s6   s180 = s1+s11
s171 = s12+s2   s176 = s12-s2   s181 = s17+s7
s172 = s8+s18   s177 = s8-s18   s182 = s13+s3
s173 = s4+s14   s178 = s4-s14   s183 = s9+s19

s184 = s5-s15   s189 = s169+s179   s194 = s169-s179
s185 = s1-s11   s190 = s170+s180   s195 = s170-s180
s186 = s17-s7   s191 = s171+s181   s196 = s171-s181
s187 = s13-s3   s192 = s172+s182   s197 = s172-s182
s188 = s9-s19   s193 = s173+s183   s198 = s173-s183

s199 = s190+s193   s200 = s190-s193   s201 = s192+s191
s202 = s192-s191   s203 = s199+s201   s204 = s199-s201
s205 = s200+s202   s206 = s203+s189
s207 = s195+s198   s208 = s195-s198   s209 = s197+s196
```

APPENDIX

```
s210 = s197-s196   s211 = s207+s209   s212 = s207-s209
s213 = s208+s210   s214 = s211+s194
s215 = s175+s178   s216 = s175-s178   s217 = s177+s176
s218 = s177-s176   s219 = s215+s217   s220 = s215-s217
s221 = s216+s218   s222 = s219+s174
s223 = s185+s188   s224 = s185-s188   s225 = s187+s186
s226 = s187-s186   s227 = s223+s225   s228 = s223-s225
s229 = s224+s226   s230 = s227+s184

m24 = (cos(w)-1)*s206
m25 = (cos(w)-1)*((cos(u)+cos(2*u))/2-1)*s203
m26 = (cos(w)-1)*((cos(u)-cos(2*u))/2)*s204
m27 = (cos(w)-1)*j*(sin(u)+sin(2*u))*s200
m28 = (cos(w)-1)*j*sin(2*u)*s205
m29 = (cos(w)-1)*j*(sin(u)-sin(2*u))*s202
m30 = (cos(w)-1)*s214
m31 = (cos(w)-1)*((cos(u)+cos(2*u))/2-1)*s211
m32 = (cos(w)-1)*((cos(u)-cos(2*u))/2)*s212
m33 = (cos(w)-1)*j*(sin(u)+sin(2*u))*s208
m34 = (cos(w)-1)*j*sin(2*u)*s213
m35 = (cos(w)-1)*j*(sin(u)-sin(2*u))*s210
m36 = (cos(w)-1)*s222
m37 = (cos(w)-1)*((cos(u)+cos(2*u))/2-1)*s219
m38 = (cos(w)-1)*((cos(u)-cos(2*u))/2)*s220
m39 = (cos(w)-1)*j*(sin(u)+sin(2*u))*s216
m40 = (cos(w)-1)*j*sin(2*u)*s221
m41 = (cos(w)-1)*j*(sin(u)-sin(2*u))*s218
m42 = (cos(w)-1)*j*sin(v)*s230
m43 = (cos(w)-1)*j*sin(v)*((cos(u)+cos(2*u))/2-1)*s227
m44 = (cos(w)-1)*j*sin(v)*((cos(u)-cos(2*u))/2)*s228
m45 = (cos(w)-1)*j*sin(v)*j*(sin(u)+sin(2*u))*s224
m46 = (cos(w)-1)*j*sin(v)*j*sin(2*u)*s229
m47 = (cos(w)-1)*j*sin(v)*j*(sin(u)-sin(2*u))*s226

s231 = m24+m25   s232 = s231+m26   s233 = s231-m26
s234 = m27-m28   s235 = m28+m29   s236 = s232+s234
s237 = s232-s234   s238 = s233+s235   s239 = s233-s235
s240 = m30+m31   s241 = s240+m32   s242 = s240-m32
s243 = m33-m34   s244 = m34+m35   s245 = s241+s243
s246 = s241-s243   s247 = s242+s244   s248 = s242-s244
```

APPENDIX

```
s249 = m36+m37   s250 = s249+m38   s251 = s249-m38
s252 = m39-m40   s253 = m40+m41   s254 = s250+s252
s255 = s250-s252   s256 = s251+s253   s257 = s251-s253
s258 = m42+m43   s259 = s258+m44   s260 = s258-m44
s261 = m45-m46   s262 = m46+m47   s263 = s259+s261
s264 = s259-s261   s265 = s260+s262   s266 = s260-s262

s267 = m36+m42   s272 = m36-m42
s268 = s254+s263   s273 = s254-s263
s269 = s256+s265   s274 = s256-s265
s270 = s257+s266   s275 = s257-s266
s271 = s255+s264   s276 = s255-s264

s277 = s20+s30   s282 = s20-s30   s287 = s25+s35
s278 = s36+s26   s283 = s36-s26   s288 = s21+s31
s279 = s32+s22   s284 = s32-s22   s289 = s37+s27
s280 = s28+s38   s285 = s28-s38   s290 = s33+s23
s281 = s24+s34   s286 = s24-s34   s291 = s29+s39

s292 = s25-s35   s297 = s277+s287   s302 = s277-s287
s293 = s21-s31   s298 = s278+s288   s303 = s278-s288
s294 = s37-s27   s299 = s279+s289   s304 = s279-s289
s295 = s33-s23   s300 = s280+s290   s305 = s280-s290
s296 = s29-s39   s301 = s281+s291   s306 = s281-s291

s307 = s298+s301   s308 = s298-s301   s309 = s300+s299
s310 = s300-s299   s311 = s307+s309   s312 = s307-s309
s313 = s308+s310   s314 = s311+s297
s315 = s303+s306   s316 = s303-s306   s317 = s305+s304
s318 = s305-s304   s319 = s315+s317   s320 = s315-s317
s321 = s316+s318   s322 = s319+s302
s323 = s283+s286   s324 = s283-s286   s325 = s285+s284
s326 = s285-s284   s327 = s323+s325   s328 = s323-s325
s329 = s324+s326   s330 = s327+s282
s331 = s293+s296   s332 = s293-s296   s333 = s295+s294
s334 = s295-s294   s335 = s331+s333   s336 = s331-s333
s337 = s332+s334   s338 = s335+s292

m48 = j*sin(w)*s314
m49 = j*sin(w)*((cos(u)+cos(2*u))/2-1)*s311
```

APPENDIX

```
m50 = j*sin(w)*((cos(u)-cos(2*u))/2)*s312
m51 = j*sin(w)*j*(sin(u)+sin(2*u))*s308
m52 = j*sin(w)*j*sin(2*u)*s313
m53 = j*sin(w)*j*(sin(u)-sin(2*u))*s310
m54 = j*sin(w)*s322
m55 = j*sin(w)*((cos(u)+cos(2*u))/2-1)*s319
m56 = j*sin(w)*((cos(u)-cos(2*u))/2)*s320
m57 = j*sin(w)*j*(sin(u)+sin(2*u))*s316
m58 = j*sin(w)*j*sin(2*u)*s321
m59 = j*sin(w)*j*(sin(u)-sin(2*u))*s318
m60 = j*sin(w)*s330
m61 = j*sin(w)*((cos(u)+cos(2*u))/2-1)*s327
m62 = j*sin(w)*((cos(u)-cos(2*u))/2)*s328
m63 = j*sin(w)*j*(sin(u)+sin(2*u))*s324
m64 = j*sin(w)*j*sin(2*u)*s329
m65 = j*sin(w)*j*(sin(u)-sin(2*u))*s326
m66 = j*sin(w)*j*sin(v)*s338
m67 = j*sin(w)*j*sin(v)*((cos(u)+cos(2*u))/2-1)*s335
m68 = j*sin(w)*j*sin(v)*((cos(u)-cos(2*u))/2)*s336
m69 = j*sin(w)*j*sin(v)*j*(sin(u)+sin(2*u))*s332
m70 = j*sin(w)*j*sin(v)*j*sin(2*u)*s337
m71 = j*sin(w)*j*sin(v)*j*(sin(u)-sin(2*u))*s334

s339 = m48+m49   s340 = s339+m50   s341 = s339-m50
s342 = m51-m52   s343 = m52+m53   s344 = s340+s342
s345 = s340-s342   s346 = s341+s343   s347 = s341-s343
s348 = m54+m55   s349 = s348+m56   s350 = s348-m56
s351 = m57-m58   s352 = m58+m59   s353 = s349+s351
s354 = s349-s351   s355 = s350+s352   s356 = s350-s352
s357 = m60+m61   s358 = s357+m62   s359 = s357-m62
s360 = m63-m64   s361 = m64+m65   s362 = s358+s360
s363 = s358-s360   s364 = s359+s361   s365 = s359-s361
s366 = m66+m67   s367 = s366+m68   s368 = s366-m68
s369 = m69-m70   s370 = m70+m71   s371 = s367+s369
s372 = s367-s369   s373 = s368+s370   s374 = s368-s370

s375 = m60+m66   s380 = m60-m66
s376 = s362+s371   s381 = s362-s371
s377 = s364+s373   s382 = s364-s373
s378 = s365+s374   s383 = s365-s374
```

APPENDIX

```
s379 = s363+s372   s384 = s363-s372

s385 = m0+m24   s405 = s385+m48   s425 = s385-m48
s386 = s160+s268   s406 = s386+s376   s426 = s386-s376
s387 = s139+s247   s407 = s387+s355   s427 = s387-s355
s388 = s167+s275   s408 = s388+s383   s428 = s388-s383
s389 = s129+s237   s409 = s389+s345   s429 = s389-s345
s390 = s159+s267   s410 = s390+s375   s430 = s390-s375
s391 = s137+s245   s411 = s391+s353   s431 = s391-s353
s392 = s166+s274   s412 = s392+s382   s432 = s392-s382
s393 = s131+s239   s413 = s393+s347   s433 = s393-s347
s394 = s163+s271   s414 = s394+s379   s434 = s394-s379
s395 = m6+m30   s415 = s395+m54   s435 = s395-m54
s396 = s165+s273   s416 = s396+s381   s436 = s396-s381
s397 = s130+s238   s417 = s397+s346   s437 = s397-s346
s398 = s162+s270   s418 = s398+s378   s438 = s398-s378
s399 = s138+s246   s419 = s399+s354   s439 = s399-s354
s400 = s164+s272   s420 = s400+s380   s440 = s400-s380
s401 = s128+s236   s421 = s401+s344   s441 = s401-s344
s402 = s161+s269   s422 = s402+s377   s442 = s402-s377
s403 = s140+s248   s423 = s403+s356   s443 = s403-s356
s404 = s168+s276   s424 = s404+s384   s444 = s404-s384

A0 = m0   A40 = s405   A20 = s425
A21 = s160   A1 = s406   A41 = s426
A42 = s139   A22 = s407   A2 = s427
A3 = s167   A43 = s408   A23 = s428
A24 = s129   A4 = s409   A44 = s429
A45 = s159   A25 = s410   A5 = s430
A6 = s137   A46 = s411   A26 = s431
A27 = s166   A7 = s412   A47 = s432
A48 = s131   A28 = s413   A8 = s433
A9 = s163   A49 = s414   A29 = s434
A30 = m6   A10 = s415   A50 = s435
A51 = s165   A31 = s416   A11 = s436
A12 = s130   A52 = s417   A32 = s437
A33 = s162   A13 = s418   A53 = s438
A54 = s138   A34 = s419   A14 = s439
A15 = s164   A55 = s420   A35 = s440
A36 = s128   A16 = s421   A56 = s441
```

APPENDIX

```
A57 = s161   A37 = s422   A17 = s442
A18 = s140   A58 = s423   A38 = s443
A39 = s168   A19 = s424   A59 = s444
```

APPENDIX

B.0 TWIDDLE FACTORS

Tables 6-13 of this appendix present the theoretical values and 12-bit approximations of the twiddle factors in the sets TF0, TF1, TF2, and TF3. Each 12-bit value is encoded in two's complement notation with a binary point after the third bit. The quantization error does not exceed 0.076%.

The bits of the twiddle factors TF1, TF2, and TF3 must be inverted before being stored into the multipliers described in this document. The bits of TF0 need not be inverted. The least significant bit must be stored into the first multiplier stage, where the multiplicands enter, and the most significant bit into the last stage, where the product exits.

APPENDIX

Table 6: Twiddle factors in TF0 (real side)..

No	Theoretical Value	Stored Value	
		12 Bits	Decimal
		MSB LSB	
0	1.00000000000000	(001000000000)	1.000000000
1	-1.25000000000000	(110110000000)	-1.250000000
2	0.55901699437495	(000100011110)	0.558593750
3	1.53884176858763	(001100010100)	1.539062500
4	0.58778525229247	(000100101101)	0.587890625
5	0.36327126400268	(000010111010)	0.363281250
6	1.00000000000000	(001000000000)	1.000000000
7	-1.25000000000000	(110110000000)	-1.250000000
8	0.55901699437495	(000100011110)	0.558593750
9	1.53884176858763	(001100010100)	1.539062500
10	0.58778525229247	(000100101101)	0.587890625
11	0.36327126400268	(000010111010)	0.363281250
12	1.00000000000000	(001000000000)	1.000000000
13	-1.25000000000000	(110110000000)	-1.250000000
14	0.55901699437495	(000100011110)	0.558593750
15	1.53884176858763	(001100010100)	1.539062500
16	0.58778525229247	(000100101101)	0.587890625
17	0.36327126400268	(000010111010)	0.363281250
18	-1.00000000000000	(111000000000)	-1.000000000
19	1.25000000000000	(001010000000)	1.250000000
20	-0.55901699437495	(111011100010)	-0.558593750
21	1.53884176858763	(001100010100)	1.539062500
22	0.58778525229247	(000100101101)	0.587890625
23	0.36327126400268	(000010111010)	0.363281250

APPENDIX

Table 7: Twiddle factors in TF0 (imaginary side)..

No	Theoretical Value	Stored Value	
		12 Bits	Decimal
		MSB LSB	
0	1.00000000000000	(001000000000)	1.000000000
1	-1.25000000000000	(110110000000)	-1.250000000
2	0.55901699437495	(000100011110)	0.558593750
3	-1.53884176858763	(110011101100)	-1.539062500
4	-0.58778525229247	(111011010011)	-0.587890625
5	-0.36327126400268	(111101000110)	-0.363281250
6	1.00000000000000	(001000000000)	1.000000000
7	-1.25000000000000	(110110000000)	-1.250000000
8	0.55901699437495	(000100011110)	0.558593750
9	-1.53884176858763	(110011101100)	-1.539062500
10	-0.58778525229247	(111011010011)	-0.587890625
11	-0.36327126400268	(111101000110)	-0.363281250
12	1.00000000000000	(001000000000)	1.000000000
13	-1.25000000000000	(110110000000)	-1.250000000
14	0.55901699437495	(000100011110)	0.558593750
15	-1.53884176858763	(110011101100)	-1.539062500
16	-0.58778525229247	(111011010011)	-0.587890625
17	-0.36327126400268	(111101000110)	-0.363281250
18	1.00000000000000	(001000000000)	1.000000000
19	-1.25000000000000	(110110000000)	-1.250000000
20	0.55901699437495	(000100011110)	0.558593750
21	1.53884176858763	(001100010100)	1.539062500
22	0.58778525229247	(000100101101)	0.587890625
23	0.36327126400268	(000010111010)	0.363281250

APPENDIX

Table 8: Twiddle factors in TF1 (real side)..

No	Theoretical Value	Stored Value	
		12 Bits	Decimal
		MSB LSB	
0	1.00000000000000	(001000000000)	1.000000000
1	-1.25000000000000	(110110000000)	-1.250000000
2	-0.55901699437495	(111011100010)	-0.558593750
3	-0.36327126400268	(111101000110)	-0.363281250
4	-0.95105651629515	(111000011001)	-0.951171875
5	1.53884176858762	(001100010100)	1.539062500
6	1.00000000000000	(001000000000)	1.000000000
7	-1.25000000000000	(110110000000)	-1.250000000
8	-0.55901699437495	(111011100010)	-0.558593750
9	-0.36327126400268	(111101000110)	-0.363281250
10	-0.95105651629515	(111000011001)	-0.951171875
11	1.53884176858762	(001100010100)	1.539062500
12	1.00000000000000	(001000000000)	1.000000000
13	-1.25000000000000	(110110000000)	-1.250000000
14	-0.55901699437495	(111011100010)	-0.558593750
15	-0.36327126400268	(111101000110)	-0.363281250
16	-0.95105651629515	(111000011001)	-0.951171875
17	1.53884176858762	(001100010100)	1.539062500
18	1.00000000000000	(001000000000)	1.000000000
19	-1.25000000000000	(110110000000)	-1.250000000
20	-0.55901699437495	(111011100010)	-0.558593750
21	0.36327126400268	(000010111010)	0.363281250
22	0.95105651629515	(000111100111)	0.951171875
23	-1.53884176858762	(110011101100)	-1.539062500

APPENDIX

Table 9: Twiddle factors in TF1 (imaginary side)..

No	Theoretical Value	Stored Value	
		12 Bits	Decimal
		MSB LSB	
0	1.00000000000000	(001000000000)	1.000000000
1	-1.25000000000000	(110110000000)	-1.250000000
2	-0.55901699437495	(111011100010)	-0.558593750
3	0.36327126400268	(000010111010)	0.363281250
4	0.95105651629515	(000111100111)	0.951171875
5	-1.53884176858762	(110011101100)	-1.539062500
6	1.00000000000000	(001000000000)	1.000000000
7	-1.25000000000000	(110110000000)	-1.250000000
8	-0.55901699437495	(111011100010)	-0.558593750
9	0.36327126400268	(000010111010)	0.363281250
10	0.95105651629515	(000111100111)	0.951171875
11	-1.53884176858762	(110011101100)	-1.539062500
12	1.00000000000000	(001000000000)	1.000000000
13	-1.25000000000000	(110110000000)	-1.250000000
14	-0.55901699437495	(111011100010)	-0.558593750
15	0.36327126400268	(000010111010)	0.363281250
16	0.95105651629515	(000111100111)	0.951171875
17	-1.53884176858762	(110011101100)	-1.539062500
18	-1.00000000000000	(111000000000)	-1.000000000
19	1.25000000000000	(001010000000)	1.250000000
20	0.55901699437495	(000100011110)	0.558593750
21	0.36327126400268	(000010111010)	0.363281250
22	0.95105651629515	(000111100111)	0.951171875
23	-1.53884176858762	(110011101100)	-1.539062500

APPENDIX

Table 10: Twiddle factors in TF2 (real side)..

No	Theoretical Value	Stored Value	
		12 Bits	Decimal
		MSB LSB	
0	-1.50000000000000	(110100000000)	-1.500000000
1	1.87500000000000	(001111000000)	1.875000000
2	0.83852549156242	(000110101101)	0.837890625
3	0.54490689600402	(000100010111)	0.544921875
4	1.42658477444273	(001011011010)	1.425781250
5	-2.30826265288144	(101101100010)	-2.308593750
6	-1.50000000000000	(110100000000)	-1.500000000
7	1.87500000000000	(001111000000)	1.875000000
8	0.83852549156242	(000110101101)	0.837890625
9	0.54490689600402	(000100010111)	0.544921875
10	1.42658477444273	(001011011010)	1.425781250
11	-2.30826265288144	(101101100010)	-2.308593750
12	-1.50000000000000	(110100000000)	-1.500000000
13	1.87500000000000	(001111000000)	1.875000000
14	0.83852549156242	(000110101101)	0.837890625
15	0.54490689600402	(000100010111)	0.544921875
16	1.42658477444273	(001011011010)	1.425781250
17	-2.30826265288144	(101101100010)	-2.308593750
18	-1.50000000000000	(110100000000)	-1.500000000
19	1.87500000000000	(001111000000)	1.875000000
20	0.83852549156242	(000110101101)	0.837890625
21	-0.54490689600402	(111011101001)	-0.544921875
22	-1.42658477444273	(110100100110)	-1.425781250
23	2.30826265288144	(010010011110)	2.308593750

APPENDIX

Table 11: Twiddle factors in TF2 (imaginary side)..

No	Theoretical Value	Stored Value	
		12 Bits	Decimal
		MSB LSB	
0	-1.50000000000000	(110100000000)	-1.500000000
1	1.87500000000000	(001111000000)	1.875000000
2	0.83852549156242	(000110101101)	0.837890625
3	-0.54490689600402	(111011101001)	-0.544921875
4	-1.42658477444273	(110100100110)	-1.425781250
5	2.30826265288144	(010010011110)	2.308593750
6	-1.50000000000000	(110100000000)	-1.500000000
7	1.87500000000000	(001111000000)	1.875000000
8	0.83852549156242	(000110101101)	0.837890625
9	-0.54490689600402	(111011101001)	-0.544921875
10	-1.42658477444273	(110100100110)	-1.425781250
11	2.30826265288144	(010010011110)	2.308593750
12	-1.50000000000000	(110100000000)	-1.500000000
13	1.87500000000000	(001111000000)	1.875000000
14	0.83852549156242	(000110101101)	0.837890625
15	-0.54490689600402	(111011101001)	-0.544921875
16	-1.42658477444273	(110100100110)	-1.425781250
17	2.30826265288144	(010010011110)	2.308593750
18	1.50000000000000	(001100000000)	1.500000000
19	-1.87500000000000	(110001000000)	-1.875000000
20	-0.83852549156242	(111001010011)	-0.837890625
21	-0.54490689600402	(111011101001)	-0.544921875
22	-1.42658477444273	(110100100110)	-1.425781250
23	2.30826265288144	(010010011110)	2.308593750

APPENDIX

Table 12: Twiddle factors in TF3 (real side)..

No	Theoretical Value	Stored Value	
		12 Bits	Decimal
		MSB LSB	
0	-0.86602540378444	(111001000101)	-0.865234375
1	1.08253175473055	(001000101010)	1.082031250
2	0.48412291827593	(000011111000)	0.484375000
3	0.31460214309120	(000010100001)	0.314453125
4	0.82363910354633	(000110100110)	0.824218750
5	-1.33267606400146	(110101010110)	-1.332031250
6	-0.86602540378444	(111001000101)	-0.865234375
7	1.08253175473055	(001000101010)	1.082031250
8	0.48412291827593	(000011111000)	0.484375000
9	0.31460214309120	(000010100001)	0.314453125
10	0.82363910354633	(000110100110)	0.824218750
11	-1.33267606400146	(110101010110)	-1.332031250
12	-0.86602540378444	(111001000101)	-0.865234375
13	1.08253175473055	(001000101010)	1.082031250
14	0.48412291827593	(000011111000)	0.484375000
15	0.31460214309120	(000010100001)	0.314453125
16	0.82363910354633	(000110100110)	0.824218750
17	-1.33267606400146	(110101010110)	-1.332031250
18	-0.86602540378444	(111001000101)	-0.865234375
19	1.08253175473055	(001000101010)	1.082031250
20	0.48412291827593	(000011111000)	0.484375000
21	-0.31460214309120	(111101011111)	-0.314453125
22	-0.82363910354633	(111001011010)	-0.824218750
23	1.33267606400146	(001010101010)	1.332031250

APPENDIX

Table 13: Twiddle factors in TF3 (imaginary side)..

No	Theoretical Value	Stored Value	
		12 Bits	Decimal
		MSB LSB	
0	0.86602540378444	(000110111011)	0.865234375
1	-1.08253175473055	(110111010110)	-1.082031250
2	-0.48412291827593	(111100001000)	-0.484375000
3	0.31460214309120	(000010100001)	0.314453125
4	0.82363910354633	(000110100110)	0.824218750
5	-1.33267606400146	(110101010110)	-1.332031250
6	0.86602540378444	(000110111011)	0.865234375
7	-1.08253175473055	(110111010110)	-1.082031250
8	-0.48412291827593	(111100001000)	-0.484375000
9	0.31460214309120	(000010100001)	0.314453125
10	0.82363910354633	(000110100110)	0.824218750
11	-1.33267606400146	(110101010110)	-1.332031250
12	0.86602540378444	(000110111011)	0.865234375
13	-1.08253175473055	(110111010110)	-1.082031250
14	-0.48412291827593	(111100001000)	-0.484375000
15	0.31460214309120	(000010100001)	0.314453125
16	0.82363910354633	(000110100110)	0.824218750
17	-1.33267606400146	(110101010110)	-1.332031250
18	-0.86602540378444	(111001000101)	-0.865234375
19	1.08253175473055	(001000101010)	1.082031250
20	0.48412291827593	(000011111000)	0.484375000
21	0.31460214309120	(000010100001)	0.314453125
22	0.82363910354633	(000110100110)	0.824218750
23	-1.33267606400146	(110101010110)	-1.332031250

APPENDIX

C.0 LOGIC SYMBOLS

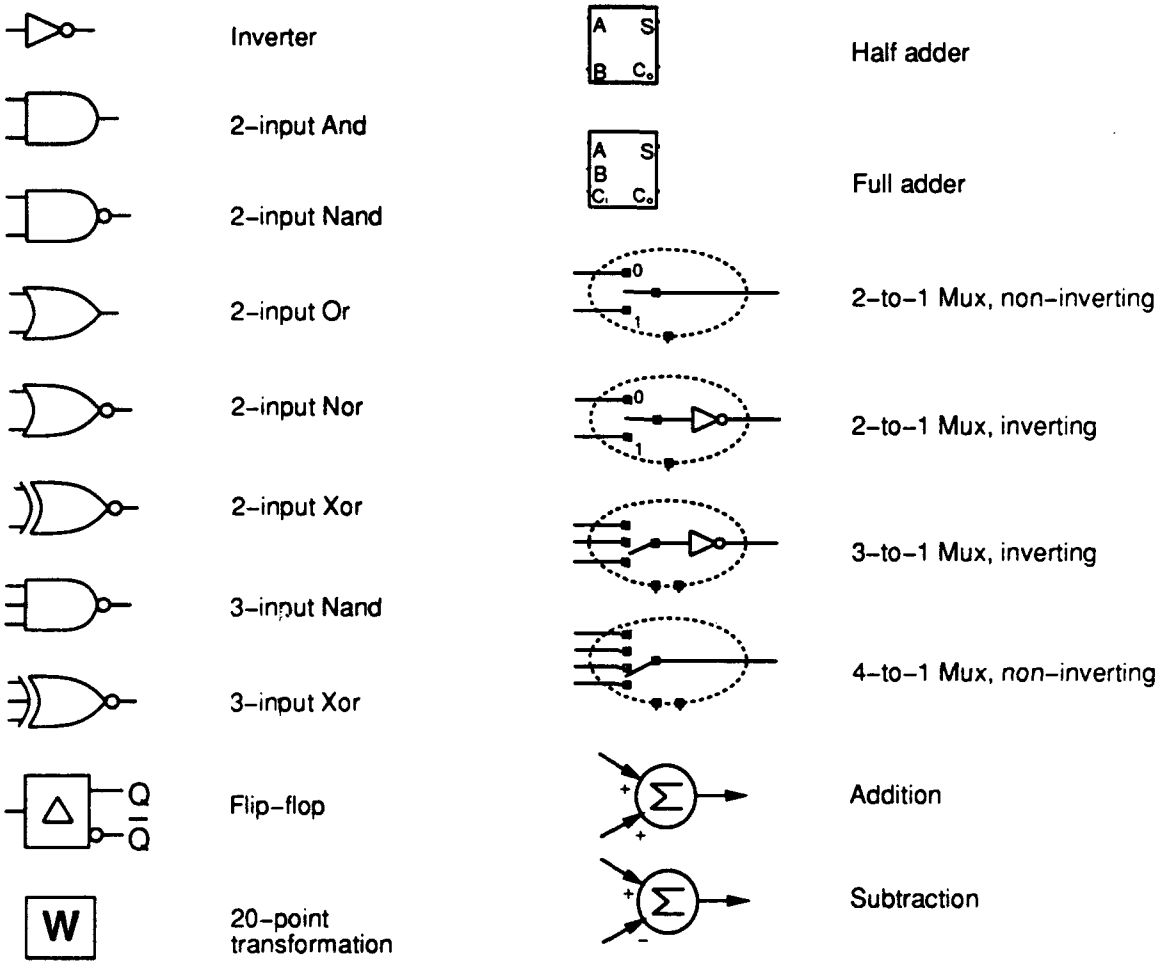


Figure 25: Logic symbols used throughout this document.

REFERENCES

[1] S. Winograd, "On computing the discrete Fourier transform," *Proc. Nat. Acad. Sci. USA*, vol. 73, no. 4, pp. 1005–1006, Apr. 1976.

[2] S. Winograd, "On computing the discrete Fourier transform," *Mathematics of Computation*, vol. 32, no. 141, pp. 175–199, Jan. 1978.

[3] R. C. Agarwal and J. W. Cooley, "New algoriths for digital convolution," *IEEE Trans. ASSP*, vol. 25, no. 5, pp. 392–410, Oct. 1977.

[4] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Mathematics of Computation*, vol. 19, no. 2, pp. 297–301, 1965.

[5] J. W. Cooley, P. A. W. Lewis, and P. D. Welch, "Historical notes on the fast Fourier transform," *IEEE Trans. Audio Electroacoust.*, vol. 15, pp. 76–79, June 1967.

[6] H. F. Silverman, "An introduction to programming the Winograd Fourier transform algorithm (WFTA)," *IEEE Trans. ASSP*, vol. 25, no. 2, pp. 152–165, Apr. 1977. "Corrections and Addendum," vol. 26, no. 3, p. 268, 1978.

[7] I. J. Good, "The interaction algorithm and practical Fourier series," *J. Royal Stat. Soc.*, vol. 20, no. 2, pp. 361–372, 1958. "Addendum," vol. 22, no. 2, pp. 372–375, 1960.

[8] L. H. Thomas, "Using a computer to solve problems in physics," in *Applications of Digital Computers*, Boston, Mass.: Ginn and Co., 1963.

[9] D. P. Kolba and T. W. Parks, "A prime factor FFT algorithm using high-speed convolution," *IEEE Trans. ASSP*, vol. 25, no. 4, pp. 281–294, Aug. 1977.

[10] R. C. Agarwal, "An in-place and in-order WFTA," in *Proc. IEEE ICASSP*, vol. 1, pp. 190–193, 1983.

[11] L. R. Morris, "A comparative study of time efficient FFT and WFTA programs for general purpose computers," *IEEE Trans. ASSP*, vol. 26, no. 2, pp. 141–150, Apr. 1978.

[12] H. Nawab and J. H. McClellan, "Bounds on the minimum number of data transfers in WFTA and FFT programs," *IEEE Trans. ASSP*, vol. 27, no. 4, pp. 394–398, Aug. 1979. "Corrections," vol. 28, no. 4, pp. 480–481, Aug. 1980.

REFERENCES

[13] R. Stasiński, "Comments on "Bounds on the minimum number of data transfers in WFTA and FFT programs"," *IEEE Trans. ASSP*, vol. 32, no. 6, pp. 1255–1257, Dec. 1984.

[14] D. Heshmaty-Manesh and S. C. Tam, "Application of Winograd's fast Fourier transform (FFT) to the calculation of diffraction optical transfer function (OFT)," in *Proc. SPIE*, vol. 369, pp. 692–695, 1983.

[15] C. Lu, J. W. Cooley, and R. Tolimieri, "Variants of the Winograd multiplicative FFT algorithms and their implementation on IBM RS/6000," in *Proc. IEEE ICASSP*, vol. 3, pp. 2185–2188, 1991.

[16] G. Aloisio, G. C. Fox, J. S. Kim, and N. Veneziani, "A concurrent implementation of the prime factor algorithm on hypercube," *IEEE Trans. Signal Proc.*, vol. 39, no. 1, pp. 160–170, Jan. 1991.

[17] R. E. Sommer and M. A. Mehalic, "Design enhancements for the air force institute of technology's Winograd Fourier transform processor," in *NAECON*, May 1991.

[18] R. W. Linderman *et al.*, "A 70-MHz 1.2-μm CMOS 16-point DFT processor," *IEEE J. Solid-State Circuits*, vol. 23, no. 2, pp. 343–350, Apr. 1988.

[19] H. J. Nussbaumer, *Fast Fourier Transform and Convolution Algorithms*. Springer-Verlag, 1981.

[20] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*. Prentice-Hall, 1989.

[21] S. Zohar, "Outline of a fast hardware implementation of Winograd's DFT algorithm," in *Proc. IEEE ICASSP*, vol. 3, pp. 796–799, 1980.

[22] J. S. Ward and B. J. Stanier, "Implementation of convolution and Fourier transforms using 1-bit systolic arrays," *Electron. Letters*, vol. 18, no. 18, pp. 799–801, 2 Sept. 1982.

[23] H. T. Kung and C. E. Leiserson, "Algorithms for VLSI processor arrays," in *Introduction to VLSI Systems* (C. Mead and L. Conway, eds.), ch. 8.3, pp. 271–292, Addison-Wesley, 1980.

REFERENCES

[24] H. T. Kung, "Why systolic architectures?," *IEEE Computer Mag.*, vol. 15, no. 1, pp. 37–46, Jan. 1982.

[25] M. D. MacLeod and N. L. Bragg, "Fast hardware implementation of the Winograd Fourier transform algorithm," *Electron. Letters*, vol. 19, no. 10, pp. 363–365, 12 May 1983.

[26] J. P. Costello, "Digital beamforming radar VLSI processor," in *Proc. SPIE*, vol. 564, pp. 66–72, 1985.

[27] J. S. Ward, J. V. McManny, and J. G. McWirter, "Bit-level systolic array implementation of the Winograd Fourier transform algorithm," *Proc. IEE (UK), Pt. F*, vol. 132, no. 6, pp. 473–479, Oct. 1985.

[28] J. S. Ward, J. V. McCanny, and J. G. McWhirter, "A systolic implmentation of the Winograd Fourier transform algorithm," in *Proc. IEEE ICASSP*, vol. 4, pp. 1469–1472, 1985.

[29] R. M. Owens and J. Ja'Ja', "A VLSI chip for the Winograd/prime factor algorithm to compute the discrete Fourier transform," *IEEE Trans. ASSP*, vol. 34, no. 4, pp. 979–989, Aug. 1986.

[30] P. Lavoie, "On the high-sped computation of short discrete Fourier transforms using the Winograd algorithm," in *Proc. of the CRAD Signal Proc. Symp. at the Defence Research Establishment Ottawa* (M. T. Rey, ed.), Nov. 1991.

[31] LSI Logic, *DSP Application Note Number One: Implementing Fast Fourier Transform Systems with the L64280/81 chip Set*, Apr. 1990.

[32] Austek Microsystems, *A User Guide for the A41102 (Preliminary)*, 1988.

[33] array Microsystems, *array Microsystems a66 Product Family Shortform Catalog*, 1991.

[34] Plessey Semiconductors, *Digital Signal Processing/ IC Handbook*, Nov. 1990.

[35] R. Meyer and K. Schwarz, "FFT implementation on DSP-chips–theory and practice," in *Proc. IEEE ICASSP*, vol. 3, pp. 1503–1506, 1990.

REFERENCES

[36] S. A. Maiorov, Y. N. Matveev, and E. F. Ochin, "Analysis of Winograd discrete Fourier transform method for 2^n-point sequences," *Automatic Contr. and Comput. Sciences*, vol. 16, no. 2, pp. 66–69, 1982. Translated from "Avtomatika i Vychislitel'naya Teknika", vol. 16, no. 2, pp. 77-80, 1982.

[37] R. Ishii and R. Watanabe, "The Winograd Fourier transform algorithm for a size of a power of 2," in *Proc. IEEE ICASSP*, vol. 2, pp. 945–948, 1985.

[38] R. Stasiński, "Rader-Winograd's DFT algorithms for $n = 2^r$," in *Proc. EUSIPCO*, vol. 1, pp. 81–84, 1986.

[39] J. S. Ward, *Number Theoretic Techniques Applied to Algorithms and Architectures for Digital Signal Processing.* PhD thesis, University of Durham, UK, Aug. 1983.

[40] R. Tolimieri, M. An, and C. Lu, *Algorithms for Discrete Fourier Transform and Convolution.* Springer-Verlag, 1989.

[41] LSI Logic, *0.7-Micron Array-Based Products Databook*, Sept. 1991.

[42] C. D. Thompson, "Fourier transform in VLSI," *IEEE Trans. Comput.*, vol. 32, no. 11, pp. 1047–1057, Nov. 1983.

[43] R. F. Lyon, "Two's complement pipeline multipliers," *IEEE Trans. Commun.*, vol. 24, no. 4, pp. 418–425, Apr. 1976.

[44] L. B. Jackson, S. F. Kaiser, and H. S. McDonald, "An approach to the implementation of digital filters," *IEEE Trans. Audio Electroacoust.*, vol. 16, pp. 413–421, Sept. 1968.

[45] L. A. Glasser and D. W. Dobberpuhl, *The Design and Analysis of VLSI Circuits.* Addison-Wesley, 1985.

[46] The Math Works Inc., *MATLAB for Sun Workstations: User's Guide*, Jan. 1990.

[47] R. W. Patterson and J. H. McClellan, "Fixed-point error analysis of Winograd Fourier transform algorithms," *IEEE Trans. ASSP*, vol. 26, no. 5, pp. 447–455, Oct. 1978.

REFERENCES

[48] N. S. Reddy and V. U. Reddy, "Implementation of Winograd's algorithm in modular arithmetic for digital convolutions," *Electron. Letters*, vol. 14, no. 7, pp. 228–229, 30 March 1978.

[49] G. Panda, R. N. Pal, and B. Chatterjee, "On the effect of correlation between truncation errors in fixed-point error analysis of Winograd short-length DFT algorithms," *IEEE Trans. ASSP*, vol. 30, no. 1, pp. 100–104, Feb. 1982.

[50] G. Panda, R. N. Pal, and B. Chatterjee, "Error analysis of Good-Winograd algorithm assuming correlated truncation errors," *IEEE Trans. ASSP*, vol. 31, no. 2, pp. 508–512, Apr. 1983.

[51] R. Stasiński and E. Lukasic, "Minimization of rounding errors in WFTA programs," in *Proc. IEEE ICASSP*, vol. 3, pp. 1423–1426, 1988.

[52] E. Lukasic, "Experimental results in mininizing errors in fixed-point WFTA programs," in *Proc. EUSIPCO*, vol. 3, pp. 1543–1546, 1990.

[53] T. W. Williams and K. P. Parker, "Design for testability—a survey," *Proc. IEEE*, vol. 71, no. 1, pp. 98–112, Jan. 1983.

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Establishment sponsoring a contractor's report, or tasking agency, are entered in section 8.)

NATIONAL DEFENCE
DEFENCE RESEARCH ESTABLISHMENT OTTAWA
SHIRLEY BAY, OTTAWA, ONTARIO K1A 0K2 CANADA

2. SECURITY CLASSIFICATION (overall security classification of the document. including special warning terms if applicable)

UNCLASSIFIED

3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C or U) in parentheses after the title.)

ON THE DESIGN OF VLSI CIRCUITS FOR THE WINOGLAD FOURIER TRANSFORM ALGORITHM (U)

4. AUTHORS (Last name, first name, middle initial)

LAVOIE, PIERRE AND MARTINEAU, SERGE

5. DATE OF PUBLICATION (month and year of publication of document)

DECEMBER 1991

6a. NO. OF PAGES (total containing information. Include Annexes, Appendices, etc.)

82

6b. NO. OF REFS (total cited in document)

53

7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

DREO REPORT

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include the address.)

NATIONAL DEFENCE
DEFENCE RESEARCH ESTABLISHMENT OTTAWA
SHIRLEY BAY, OTTAWA, ONTARIO K1A 0K2 CANADA

9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant)

011LB11

9b. CONTRACT NO. (if appropriate, the applicable number under which the document was written)

10a. ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique to this document.)

DREO REPORT 1108

10b. OTHER DOCUMENT NOS. (Any other numbers which may be assigned this document either by the originator or by the sponsor)

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)

- (X) Unlimited distribution
- ( ) Distribution limited to defence departments and defence contractors; further distribution only as approved
- ( ) Distribution limited to defence departments and Canadian defence contractors; further distribution only as approved
- ( ) Distribution limited to government departments and agencies; further distribution only as approved
- ( ) Distribution limited to defence departments; further distribution only as approved
- ( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.)

UNCLASSIFIED

SECURITY CLASSIFICATION OF FORM

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both offical languages unless the text is bilingual).

(U) A VLSI architecture for computing the discrete Fourier transform (DFT) using the Winograd Fourier transform algorithm (WFTA) is presented. This architecture is an addressless, routed, bit-serial scheme that directly maps an N-point algorithm onto silicon. The architecture appears to be far less costly than systolic schemes for implementing the WFTA, and faster than current FFT devices for similar transform sizes. The nesting method of Winograd is used for partitioning larger transformations into several circuits. The advantage of tis partitioning technique is that it allows using circuits that are all of the same type. However, the number of input/output pins of each circuit is higher than with some other approaches like, for example, the prime factor algorithm. The design of a 20-point DFT circuit with logic diagrams of its major major cells is presented. The gate array circuit has been sent for fabrication in a 0.7 um CMOS technology. Five circuits interconnected together will compute 60-point complex transforms at a rate of one transformation every 0.53 us.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

VLSI ARCHITECTURE
DFT
DISCRETE FOURIER TRANSFORM
WFTA
WINOGRAD
HARDWARE
INTEGRATED CIRCUIT
NESTING METHOD

UNCLASSIFIED

SECURITY CLASSIFICATION OF FORM